DTIC FILE COPY

CURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS NONE
AD-A217 924	3. DISTRIBUTION/AVAILABILITY OF REPORT APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.
S)	5. MONITORING ORGANIZATION REPORT NUMBER(S) AFIT/CI/CIA-89-028

6a. NAME OF PERFORMING ORGANIZATION AFIT STUDENT AT UNIVof OK	6b. OFFICE SYMBOL (If applicable)	7a. NAME OF MONITORING ORGANIZATION AFIT/CIA
6c. ADDRESS (City, State, and ZIP Code)		7b. ADDRESS (City, State, and ZIP Code) Wright-Patterson AFB OH 45433-6583
8a. NAME OF FUNDING/SPONSORING ORGANIZATION	8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State, and ZIP Code)		10. SOURCE OF FUNDING NUMBERS

PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO.	WORK UNIT ACCESSION NO.

11. TITLE (Include Security Classification) (UNCLASSIFIED)
DEVELOPMENT OF COMPUTER SOFTWARE FOR THE ANALYSIS AND DESIGN OF MODERN CONTROL SYSTEMS

12. PERSONAL AUTHOR(S)
CARL FRED ADAMS

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
THESIS/~~DISSERTATION~~	FROM _____ TO _____	1989	116

16. SUPPLEMENTARY NOTATION
APPROVED FOR PUBLIC RELEASE IAW AFR 190-1
ERNEST A. HAYGOOD, 1st Lt, USAF
Executive Officer, Civilian Institution Programs

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

DTIC
ELECTE
FEB 12 1990
S D
D

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED
22a. NAME OF RESPONSIBLE INDIVIDUAL ERNEST A. HAYGOOD, 1st Lt, USAF	22b. TELEPHONE (Include Area Code) (513) 255-2259
	22c. OFFICE SYMBOL AFIT/CI

DD Form 1473, JUN 86 *Previous editions are obsolete.* SECURITY CLASSIFICATION OF THIS PAGE

AFIT/CI "OVERPRINT"

28

THE UNIVERSITY OF OKLAHOMA

GRADUATE COLLEGE

DEVELOPMENT OF COMPUTER SOFTWARE FOR THE ANALYSIS AND DESIGN OF MODERN CONTROL SYSTEMS

A THESIS

SUBMITTED TO THE GRADUATE FACULTY

in partial fulfillment of the requirements for the

degree of

MASTER OF SCIENCE

By

CARL FRED ADAMS

Norman, Oklahoma

1989

DEVELOPMENT OF COMPUTER SOFTWARE FOR THE ANALYSIS AND DESIGN OF MODERN CONTROL SYSTEMS

A THESIS

APPROVED FOR THE SCHOOL OF AEROSPACE AND MECHANICAL ENGINEERING

By

Dawn M. Eglen

William Neff Patten

Robert J. Whitehead

ACKNOWLEDGMENTS

The author would like to thank his advisor, Dr. Davis M. Egle, for his guidance, support, and encouragement while at the University of Oklahoma. Also, the author would like to express his appreciation to his committee members, Dr. Robert J. Mulholland and William N. Patten for their expert advice.

The United States Air Force and the Air Force Institute of Technology is acknowledged for providing the opportunity and financial support for the author to study at the University of Oklahoma.

I dedicate this thesis to my wife Karen. It was her encouragement, strength, and love that has made this effort worthwhile.

TABLE OF CONTENTS

LIST OF FIGURES

GLOSSARY OF SYMBOLS

c(s)	system output
cg	centroid or center of gravity
G(s)	forward transfer function
H(s)	feedback transfer function
K	close-loop gain
L_1, L_2, L_3	right angle test data points
M	number of transfer function poles
m	root locus slope
M_r	maximum Bode magnitude value
N	number of transfer function zeros
n_p	number of transfer function poles
n_z	number of transfer function zeros
omegal	lower value of Bode frequency range
omegau	upper value of Bode frequency range
p_i	transfer function pole
r(s)	system input
S_Q	initial s-plane root locus data point
S_R	final s-plane root locus data point
T(s)	close-loop transfer function
X_k	recursive formula for real part of the transfer function T(s)
Y_k	recursive formula for imaginary part of the transfer function T(s)
z_i	transfer function zero
α_i	angle criterion angle
α_d	root locus departure angle
θ_k	asymptotic root locus convergence angle
ξ	damping coefficient
$\phi(\omega)$	Bode phase angle
ω	frequency
ω_r	resonant frequency

DEVELOPMENT OF COMPUTER SOFTWARE FOR THE ANALYSIS AND DESIGN OF MODERN CONTROL SYSTEMS

CHAPTER 1

PRELIMINARIES

1.1 Introduction to the Control Systems Software Package

The classical techniques of graphical analysis and design of modern control systems involve the generation of sketches by hand. In all but the most simple systems this can be a tedious and difficult process. The purpose of this research was to develop computer software that automates this task. The software programs developed include the Root Locus, Bode Plot and all necessary support routines needed to fully automate the development of these graphs. This software is called collectively the *Control Systems Software Package* (CSSP).

1.2 Computer System Requirements for CSSP

The CSSP programs are written for use on IBM and compatible personal computers with a graphics card and monitor installed. On machines with an Enhanced Graphics Adaptor (EGA), or higher resolution monitors, the graphs are produced in color. Computers with a Color Graphics Adaptor

(CGA) are limited to black and white plots due to the low resolution of the monitor when plotting in color.

A printer capable of supporting a graphics screen dump is required if a printout is necessary, but the printer isn't specifically needed to operate the program. A numeric coprocessor (8087, 80287 or 80387) is not required, but is highly recommended. This coprocessor increases computational speed considerably. All hardware specific settings are totally automatic and are invisible to the user.

1.3 The C Computer Language

All of CSSP's source code is written in the C language and is included for future study and possible refinement. Since the C language is very potable and is not tied to any particular hardware or operating system, very little effort would be required to rewrite this software to operate on other types computer systems. This is true for all but the video specific functions, such as the windowing routines.

The CSSP source code, provided in the Appendix, was written for and compiled with the Micorsoft C Compiler version 5.0. The windowing software used to develop the screens and input windows was "The Window BOSS" Revision : 08.15.88 written by Phil Mongelluzzo of Star Guidance Consulting, Inc.. All the necessary source code and libraries needed to reproduce CSSP are provided on a 5 1/4 inch diskette, which is available at the Aerospace and Mechanical Engineering office, University of Oklahoma.

This source code is provided for educational purposes only, and is not to be used in any type of commercial application without the prior

1.4 Thesis Overview

The remainder of the thesis is devoted to the discussion of the classical techniques of analysis and design using the Root Locus and Bode plots. Along with these techniques will be a discussion of the equivalent computer-aided techniques used in the CSSP program. The following is a short description of the remaining chapters.

Chapter 2: The Root Locus Method. This chapter discusses the Root Locus method of describing the transient response of a control system by determining the location of the roots of the characteristic equation as the open-loop gain is varied from zero to infinity. This chapter also introduces a set of ten rules used to help in the plotting of root locus by hand.

Chapter 3: Computer-Aided Root Locus Plotting. This chapter introduces the two most common methods of computer-aided root locus plotting. They are the numerical polynomial-factoring and angle testing methods. It talks about why the polynomial-factoring method is the simplest algorithm to program, but the price of simplicity is a longer computer run time. It also explains why the angle testing method is the harder of the two methods to program, but is usually the more efficient method for computer run time.

Chapter 4: Root Locus Program. This chapter discusses the root locus program used in the CSSP. It uses the angle testing method for plotting.

Chapter 5: Bode Plots. This chapter serves as an introduction to Bode Plots. This is a graphical technique that plots a control system's gain amplitude and phase angle response curves against the input frequency.

Chapter 6: Computer-Aided Bode PLots. This chapter describes the Bode plot algorithm used in the CSSP.

Chapter 7: Summary. This is the thesis summary.

CHAPTER 2

THE ROOT LOCUS METHOD

2.1 Introduction to the Root Locus Method

The root locus method is a powerful graphical tool used in the analysis and design of linear control systems. This method determines the location of the roots of the characteristic equation as the open-loop gain is varied from zero to infinity. By knowing the locations of these roots it is possible to not only know if a system is stable but also the degree of its stability. This method is another way of describing the transient response of the system. If a system is unstable or has undesirable response characteristics the root locus method provides possible qualitative ways to improve it.

2.2 The Rool Locus Method

Consider the simple single-input single-output, closed-loop system shown in Figure 2.1.

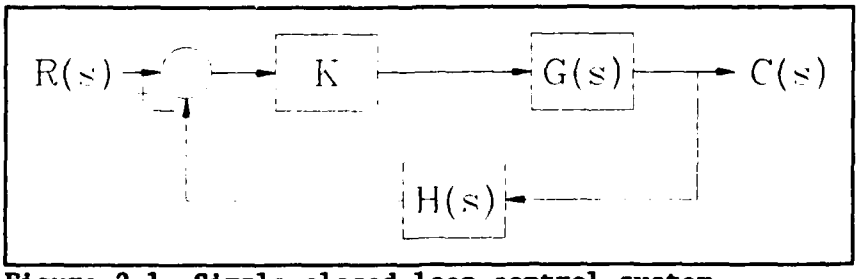

Figure 2-1 Simple closed-loop control system

The transfer function of this system is

$$T(s) = \frac{C(s)}{R(s)} = \frac{KG(s)}{1+KG(s)H(s)} \tag{2.1}$$

and the characteristic equation of this system is obtained by setting the denominator equal to zero. Thus the general form of the characteristic equation is given by

$$1 + KG(s)H(s) = 0 \quad \text{or} \quad KG(s)H(s) = -1 \tag{2.2}$$

or, in polar form,

$$KG(s)H(s) = 1\angle 180^\circ \tag{2.3}$$

where K is the gain of the system. The characteristic equation contains all the information about a system's frequency response, time response, and stability. In other words, any value of s that causes the open-loop transfer function G(s)H(s) to have a gain of unity and a phase shift of 180°, and by definition a point on the root locus, will be a pole of the closed-loop system. This is a very useful fact in that evaluation of the usually simpler open-loop transfer function tells how the poles of the closed-loop system behave. Therefore, by evaluating G(s)H(s) in the complex s domain and finding the values of s that make equation (2.3) true will determine how the closed-loop poles are influenced by changes in any of the system parameters.

To determine if a given value of s is a point on the root locus for some value of K between zero and $+\infty$, it is only necessary to determine whether or not the angle of KZ(s)/P(s)=KG(s)H(s) is an odd multiple of 180° and the magnitude must be equal to unity. This is known as the angle criterion and the magnitude criterion respectively and are shown in

equations (2.4) and (2.5a).

$$\sum \Phi_l = \sum(\text{angles of the zeros}) - \sum(\text{angles of the poles}) = (2k+1)180^\circ,\ k=1,2,\ldots \tag{2.4}$$

$$\frac{K\ (s-z_1)\ (s-z_2)\ \cdots}{(s-p_1)\ (s-p_2)\ \cdots} = 1 \tag{2.5a}$$

which gives

$$K = \frac{\text{product of the pole distances}}{\text{product of the zero distances}} = \frac{|P(s)|}{|Z(s)|} \tag{2.5b}$$

2.3 Rules for Plotting a Rool Locus

To help in the plotting of root loci a set of conventional rules have been developed. These rules have been thoroughly discussed in [1]. Here, a brief description of these rules is presented and applied to two systems with the following open-loop functions,

example 1: $$\frac{K}{s(s+5)(s+7)}$$

example 2: $$\frac{K(s+6)}{s(s+4)(s^2+4s+8)}$$

Root locus plots of the above open-loop functions are shown in Figures 2.2 and 2.3.

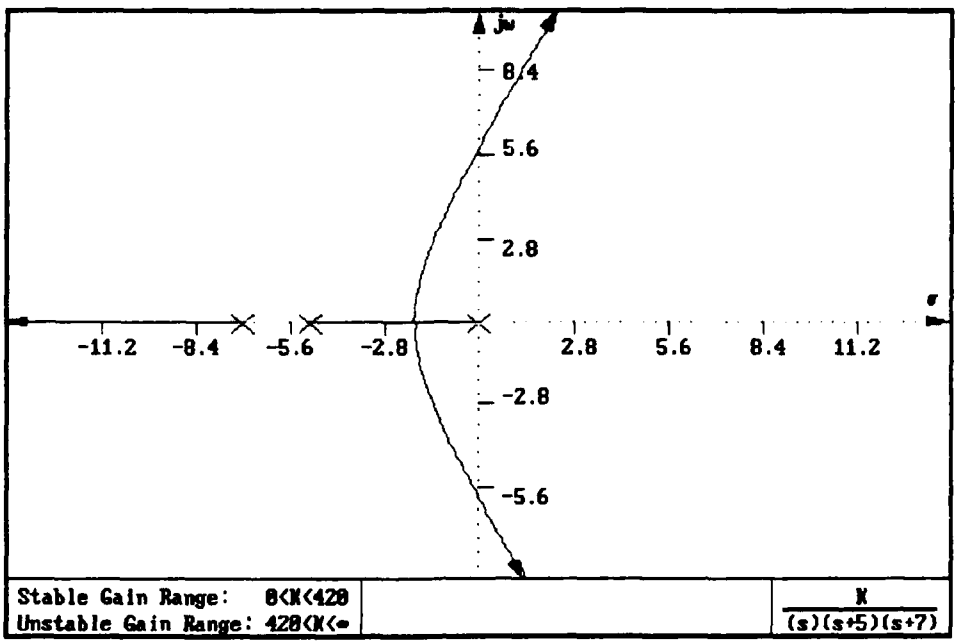

Figure 2.2 Root locus plot, example 1

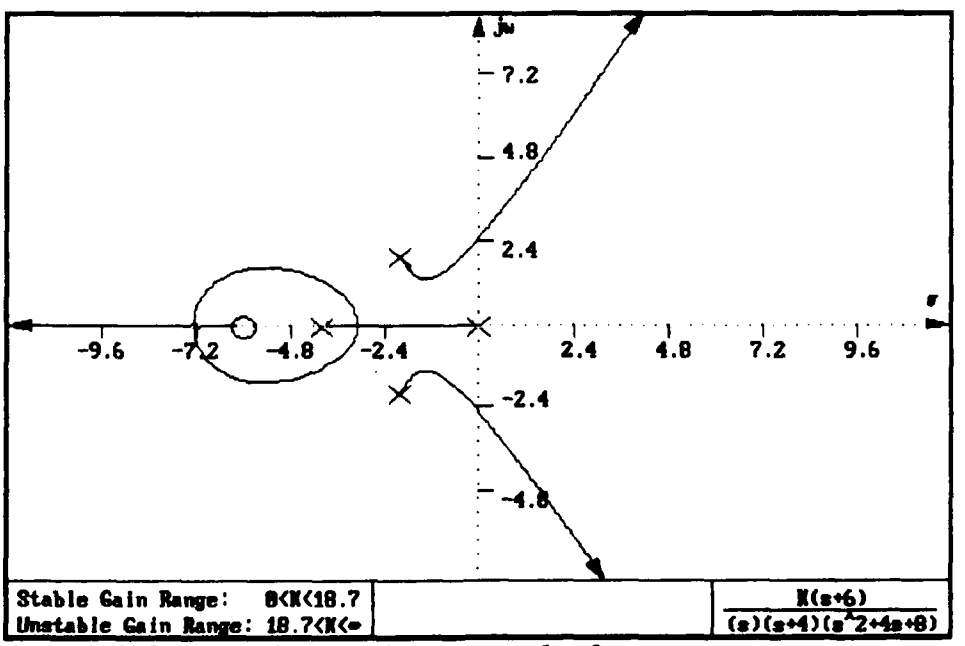

Figure 2.3 Root locus plot, example 2

RULE 1: *There is a single-valued branch of the root locus for each of the characteristic equation roots and the total number of branches is equal to the number of open-loop poles.*

Example 1: There are no zeros and three poles, therefore; there is a total of three locus branches.

Example 2: There is one zero and four poles, therefore; there is a total of four locus branches.

RULE 2: *Each branch of the root locus starts at a pole, where K=0, and ends at a zero, where K = +∞. If the number of poles exceeds the number of zeros, there will be zeros at infinity equal in number to the excess. Excess zeros similarly mean poles at infinity.*

Example 1: There are three more poles than zeros, which means all three of the branches go to infinity.

Example 2: There are three more poles than zeros, which means three of the four branches go to infinity.

RULE 3: *Along the real axis the locus includes all points to the left of an odd number of real poles and zeros: no distinction is made between poles and zeros, and complex poles and zeros are neglected.*

Example 1: There will be points on the real axis from $0>\sigma>-5$ and from $-7>\sigma>-\infty$.

Example 2: There will be points on the real axis from $0>\sigma>-4$ and from $-6>\sigma>-\infty$.

RULE 4: *If the number of poles n_p exceeds the number of zeros n_z, then as K approaches infinity, $(n_p - n_z)$ branches will become asymptotic to straight lines intersecting the real axis at angles given by*

$$\theta_k = \frac{(2k + 1)180^\circ}{(n_p - n_z)}, \quad k = 0, 1, 2... \tag{2.6}$$

If n_z exceeds n_p, then as K approaches zero, $(n_z - n_p)$ branches behave as above.

Example 1: $$\theta_k = \frac{(2k + 1)180^\circ}{(3 - 0)} = (2k + 1)60^\circ, \quad k = 0, 1, 2$$

Example 2: $$\theta_k = \frac{(2k + 1)180^\circ}{(4 - 1)} = (2k + 1)60^\circ, \quad k = 0, 1, 2$$

In both examples there will be three asymptotes, one for each infinite zero, intersecting the real axis at angles of 60°, 180° and 300°.

RULE 5: *The asymptotes of Rule 4 will intersect the real axis at a point, called the center of gravity, given by*

$$cg = \frac{\sum \text{poles} - \sum \text{zeros}}{(n_p - n_z)} \tag{2.8}$$

Example 1: $$cg = \frac{-0 - 5 - 7}{(3 - 0)} = -4$$

Example 2: $$cg = \frac{-0 - 4 - 2 + 2j - 2 - 2j + 6}{(4 - 1)} = \frac{-2}{3}$$

RULE 6a: *A breakaway or arrival point s_b on a real axis can be found from the equality*

$$\frac{-1}{s_b + p_l} + \frac{1}{s_b + z_l} + \cdots \pm \frac{2(s_b - \alpha)}{(s_b - \alpha)^2 + \omega^2}$$

$$= \frac{-1}{s_b + p_r} + \frac{1}{s_b + z_r} + \cdots \mp \frac{2(s_b - \alpha)}{(s_b - \alpha)^2 + \omega^2} \qquad (2.9)$$

where the terms on the left side of the equation come from the poles $-p_l$ and zeros $-z_l$ to the left of the breakaway point; the terms on the right-hand side come from the poles $-p_r$ and zeros $-z_r$ to the right. Positive signs are associated with the zeros and negative signs with the poles.

RULE 6b: *A breakaway point occurs when K is at a relative maximum; an arrival point occurs when K is at a relative minimum.*

The breakaway point can be found by solving the characteristic equation (2.2) for K and then taking the derivative with respect to s and setting it equal to zero. Then solve for the resulting roots.

Example 1: The roots of (2.2) are s=-6.1, -1.9. Since the point s=-6.1 is not on the root locus, see Rule 3, the breakaway point must occur at s=-1.9.

Example 2: The two non-complex roots of (2.2) are -3.1, and -7.3. The breakaway point is approximately -3.1 and the breakin point approximately -7.3.

RULE 7: *Branches of the root locus are symmetrical with respect to the real axis since all complex roots appear in conjugate pairs.*

This implies all root locus points that lie above the real axis will have matching mirrored points below the axis.

RULE 8: *Two branches of a root locus breakaway from or arrive at the real axis at angles of ±90°.*

For three or more branches use Rule 9 below to calculate the departure or arrival angles.

Example 1: The breakaway point at s=-1.9 breaks from the real axis at $\pm 90^{\circ}$.

Example 2: The breakaway point at s=-3.1 and the breakin point at s=-7.3 break from the real axis at $\pm 90^{\circ}$.

RULE 9: *The angle criterion can be used to find the angle of departure from a complex pole and the angle of arrival to a complex zero by selecting a trial point so close to the complex pole or zero that the angles from the other poles and zeros are unaffected.*

Example 1: There are no complex poles or zeros for this example; therefore, this rule is not applicable.

Example 2: Figure 2-4 and equation 2-10 illustrate how the departure angle of -63.4°, for the pole at the point (-2+2j), is derived.

$$-\alpha_4-\alpha_1-\alpha_2-\alpha_3+\alpha_4 = 180^{\circ} \qquad (2.10)$$

RULE 10: *If $n_p-n_z\geq 2$, then the sum of the roots of the characteristic equation is constant and equal to the sum of the poles.*

This rule is limited in use, but might prove to be useful under certain circumstances. It isn't needed for examples 1 or 2. Refer to [1] for an example using this rule.

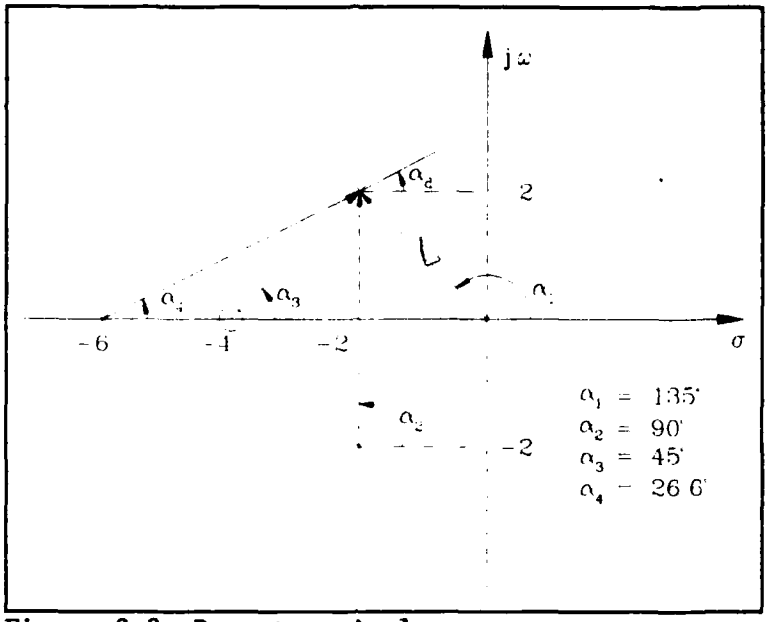

Figure 2.3: Departure Angle

CHAPTER 3

COMPUTER-AIDED ROOT LOCUS PLOTTING

3.1 Introduction to Computer-Aided Root Locus Plotting

Using the ten rules explained in Chapter 2 it is possible to construct root locus plots with a reasonable amount of accuracy; however, since this method involves hand construction and the use of a calculator, it can become confusing and tedious. To improve accuracy and to make better use of a system designer's time, a computer-aided approach to root locus plotting was developed. This chapter introduces the development of the numerical polynomial-factoring and angle testing methods used in computer-aided root locus plotting. This chapter also explains why angle testing, which is used in the root locus program discussed in chapter 4, is usually the optimum method to use for all but the simplest systems.

3.2 Numerical Polynomial Factoring

One method of computer-aided root locus plotting is to repeatedly factor the equation

$$1 + G(s)H(s) = 0 \qquad (3.1)$$

as K is varied in step increments from zero to infinity. Figure 3.1 shows a simplified programming flow chart for a program using the numerical polynomial-factoring method.

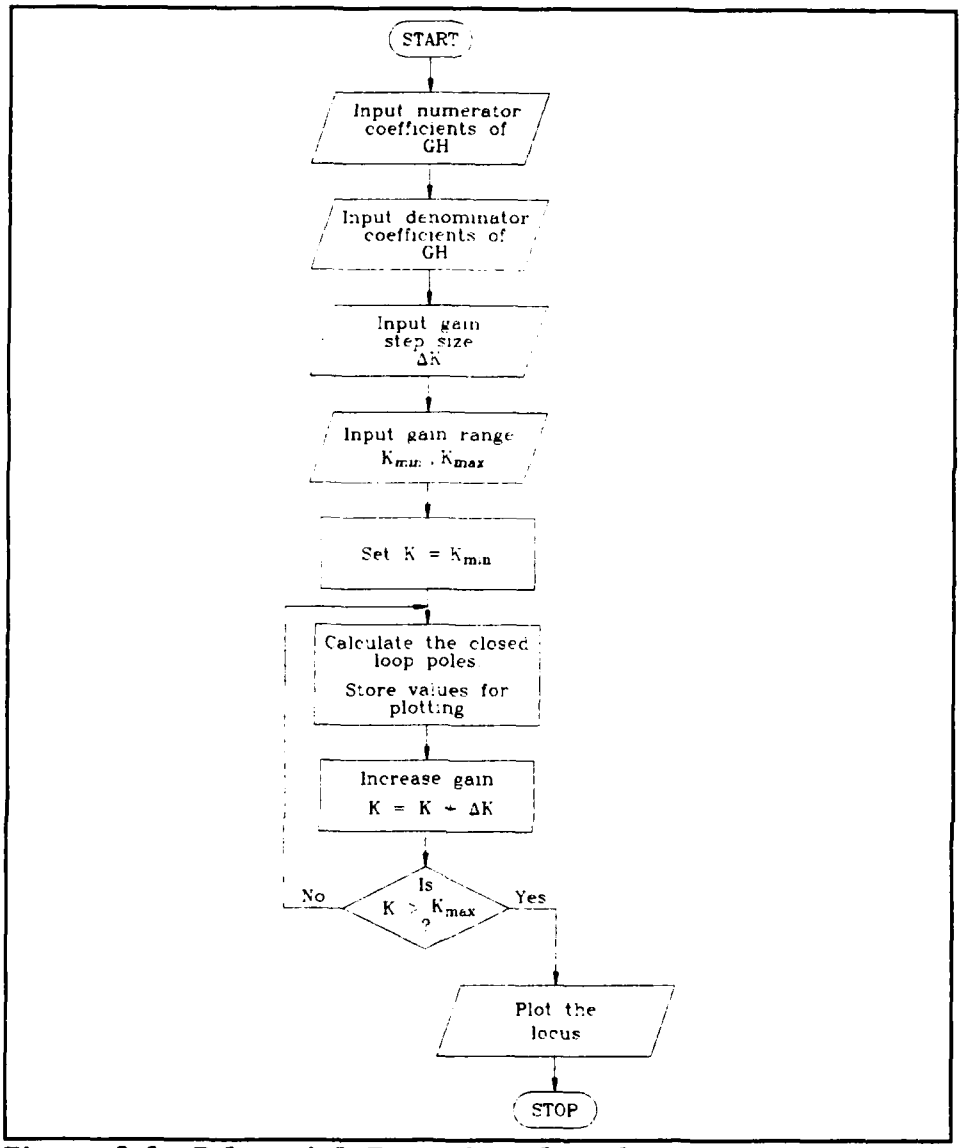

Figure 3-1 Polynomial-Factoring Flow Chart

A root of equation (3.1) is any value of $s=x+jy$ that makes

$$D(s) = 1+KG(s)H(s) = 0 \qquad (3.2)$$

true for any fixed value of K. This will only be true if and only if both the real and imaginary parts of D(s) are equal to zero.

That is

$$\begin{cases} \mathrm{Re}[D(s=x+jy)] = 0 \\ \mathrm{Im}[D(s=x+jy)] = 0 \end{cases} \tag{3.3}$$

or alternatively if

$$|D(x+jy)| = 0 \tag{3.4}$$

The programing needed to solve (3.3) or (3.4) is simple, but using this method to locate the polynomial roots is a time consuming approach. This is a two-dimensional problem in x and y and repeatedly factoring this type of polynomial can be a lengthy process. This method is especially inefficient when the gain factor, K, becomes very large. It's for this reason that the angle testing method was developed.

3.3 Method of Angle Testing

Points $s=x+jy$ on a root locus are those for which the angle

$$\angle GH(s=x+jy) = 180^\circ \tag{3.5}$$

or

$$\mathrm{Im}[D(s=x+jy)] = 0 \tag{3.6}$$

The angle testing method involves using (3.5) to find a locus departure angle starting from one of the open-loop poles. Finding the open-loop poles and zeros requires only one pass of the polynomial root finding routine, when K = 0, instead of the multiple passes of the polynomial-factoring method.

The departure angle is used as the starting point for a process that iterates the angle of G(s)H(s) until the next point is sufficiently close to satisfy (3.6) to within some given tolerance. This means that the two-dimensional problem of x and y is reduced to a one-dimensional angle

problem. This process continues along one locus until the plot limits are exceeded. Then, starting at a new open-loop pole, the whole process is started over until all loci are complete.

The program development needed to solve (3.5) and (3.6) is much more involved than that of the polynomial factoring method; however, the savings in computation time between the two methods can more than make up for this difference. This is true for all but the simplest problems. Figure 3.2 shows a simplified programming flow chart for a program using the angle testing method.

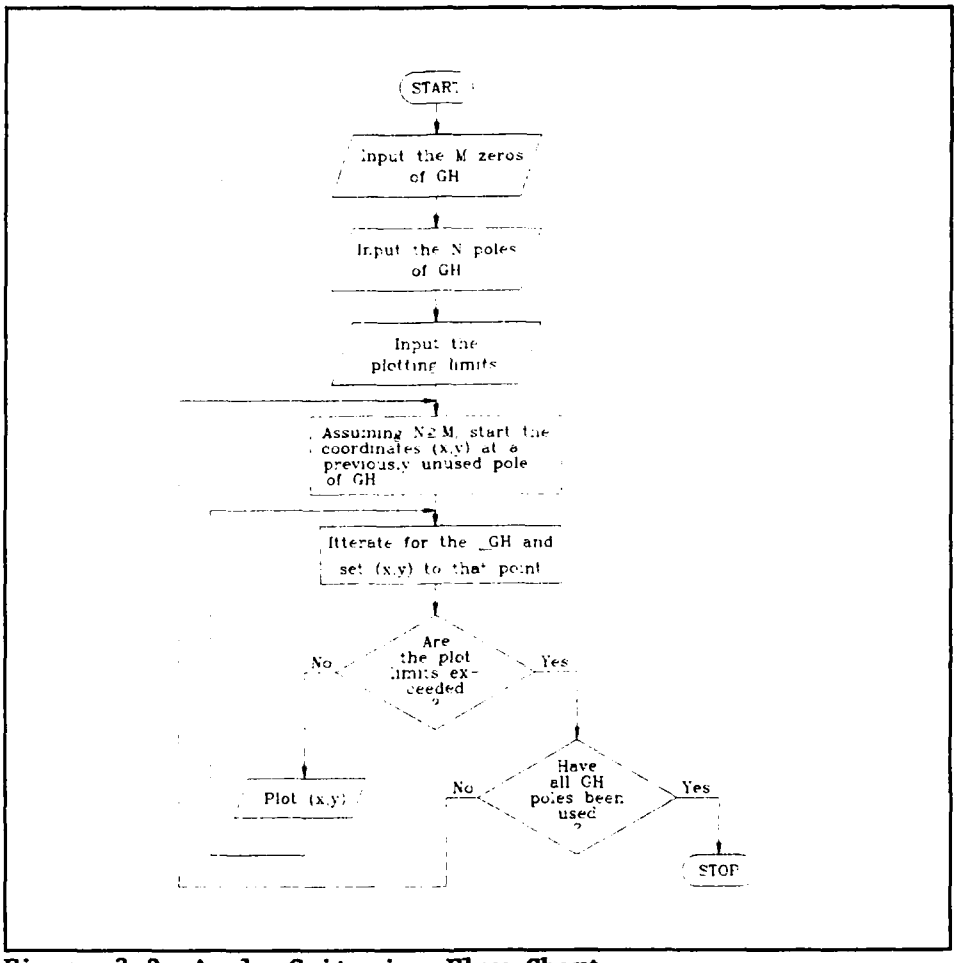

Figure 3-2 Angle Criterion Flow Chart

CHAPTER 4

ROOT LOCUS PROGRAM

4.1 Introduction to Root Locus Program

This chapter discusses the root locus plotting program which was developed to aid the control system designer in the design and analysis of control systems. The algorithm described in this chapter follows the work of J.A. Borrie [2]. Borrie's work was based on the original work done by P. Atkinson, and V.S. Dalvi [5]. This program uses the angle testing method as explained in chapter 3. This computer program is written in the C language and all source code is provided in Appendix A.

4.2 Algorithm Development

To start, it is helpful to write the system open-loop transfer function in the form

$$G(s) = G(\sigma + j\omega) \tag{4.1}$$

$$= \frac{K\prod_{i=1}^{M}[(\sigma-\sigma_{zi})+j(\omega-\omega_{zi})]}{\prod_{k=1}^{N}[(\sigma-\sigma_{pk})+j(\omega-\omega_{pk})]} \tag{4.2}$$

where

K = closed-loop gain
M = number of complex zeros
N = number of complex poles
$(\sigma_{zi}+j\omega_{zi})$ = complex zeros
$(\sigma_{pk}+j\omega_{pk})$ = complex poles

Equation (4.2) can be written in the form

$$G(\sigma - j\omega) = \frac{K\prod_{i=1}^{M}[(\sigma-\sigma_{zi}) - j(\omega-\omega_{zi})]\prod_{k=1}^{N}[(\sigma-\sigma_{pk}) - j(\omega-\omega_{pk})]}{\prod_{k=1}^{N}[(\sigma-\sigma_{pk})^2 + (\omega-\omega_{pk})^2]} \tag{4.3}$$

Simplify (4.3) by writing it in the form

$$G(\sigma - j\omega) = K[X - jY] \tag{4.4}$$

Where X is the real part and Y is the complex part of (4.3). To evaluate (4.4), start with the first pole ($\sigma_{p1}+j\omega_{p1}$),

$$X_1 - jY_1 = \frac{(\sigma-\sigma_{p1}) - j(\omega-\omega_{p1})}{(\sigma-\sigma_{p1})^2 + (\omega-\omega_{p1})^2} \tag{4.5}$$

Finish evaluating the remaining poles using the recursive formulas

$$X_k = \frac{(\sigma-\sigma_{pk})X_{k-1} + (\omega-\omega_{pk})Y_{k-1}}{(\sigma-\sigma_{pk})^2 + (\omega-\omega_{pk})^2} \tag{4.6}$$

$$Y_k = \frac{(\sigma-\sigma_{pk})Y_{k-1} - (\omega-\omega_{pk})X_{k-1}}{(\sigma-\sigma_{pk})^2 + (\omega-\omega_{pk})^2} \tag{4.7}$$

$$k = 2,3,\ldots N$$

Evaluate the zeros of (4.4) using the formulas

$$X_{i+N} = (\sigma-\sigma_{zi})X_{i+N-1} - (\omega-\omega_{zi})Y_{i+N-1} \tag{4.8}$$

$$Y_{i+N} = (\sigma-\sigma_{zi})Y_{i+N-1} - (\omega-\omega_{zi})X_{i+N-1} \tag{4.9}$$

$$i = 1,2,\ldots M$$

This gives

$$G(s) = K[X_{M+N} + jY_{M+N}] \tag{4.10}$$

Notice that equations (4.6) and (4.7) fail when the point ($\sigma+j\omega$) is near a pole. This is overcome by having the program check for, and avoid

evaluating, G(s) at these points.

Figure 4.1 shows a computer flow chart of the root locus algorithm. In the interests of simplicity, some of the less significant steps are not shown. For further details see Appendix C.

BLOCK 1 Input the zeros and poles of the transfer function.

BLOCK 2 Display the poles and zeros. The pole locations are marked with an x and the zeros are marked with a circle.

BLOCK 3 Set the loop counter, locus, to 1 and establish the allowable locus deviation to 0.00001. The error value was established through a trial and error process and is a compromise between locus accuracy and computation time.

BLOCK 4 Check the loop counter for being less than or equal to the total number of system loci.

BLOCK 5 As explained in Chapter 2, each locus starts at a pole and ends at either a zero or asymptotically at infinity. Use the angle criterion to calculate the departure angle at the start of the present pole. The angle criterion is best demonstrated by example. Consider a system with the following transfer function

$$G(s) = \frac{(s+z_1)}{(s+p_1)(s+p_2)(s+p_3)(s+p_4)} \tag{4.11}$$

As shown in Chapter 2, the angle criterion is given by

$$\sum \alpha_i = \sum(\text{angles of the zeros}) - \sum(\text{angles of the poles}) = (2k+1)\pi \tag{4.12}$$

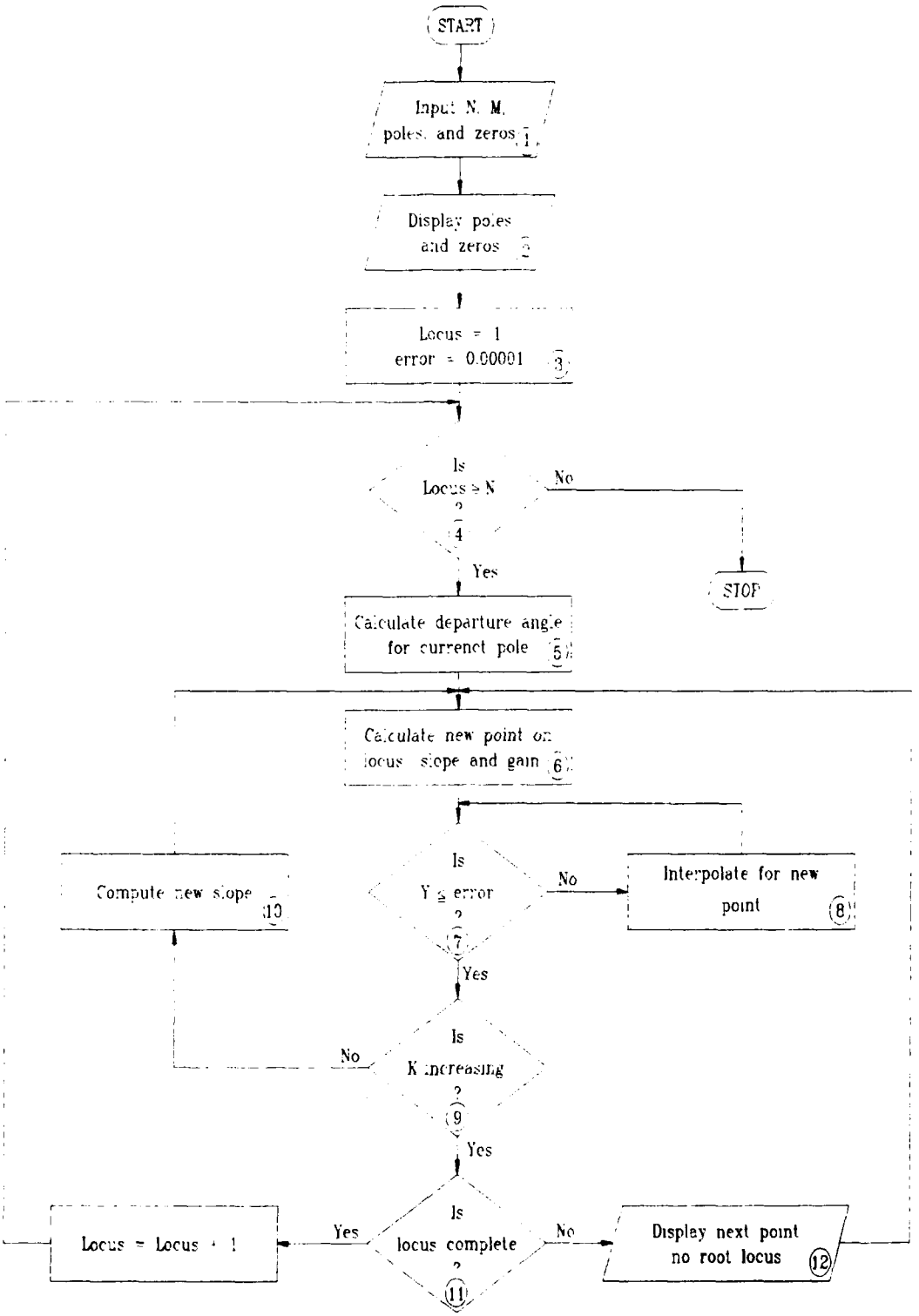

Figure 4.1 Root Locus Flow Chart

where

$$k = \pm 1, \pm 2, \pm 3, \ldots \tag{4.13}$$

is chosen so that

$$-\pi \leq \alpha_d \leq +\pi \tag{4.14}$$

where the angles of zeros and poles are shown in Figure 4.2. This gives

$$\alpha_d = \alpha_3 - \alpha_1 - \alpha_2 - \alpha_4 - (2k+1)\pi \tag{4.15}$$

BLOCK 6 To find the point S_Q near the locus, take an incremental step, with a length of Δs, from the pole S_P in the direction of α_d. This can be seen in figure 4.3. Δs is computed to be approximately two percent of the largest open-loop pole or zero coordinate.

$$S_Q = S_P + \Delta s(\cos\alpha_d + j\sin\alpha_d) \tag{4.16}$$

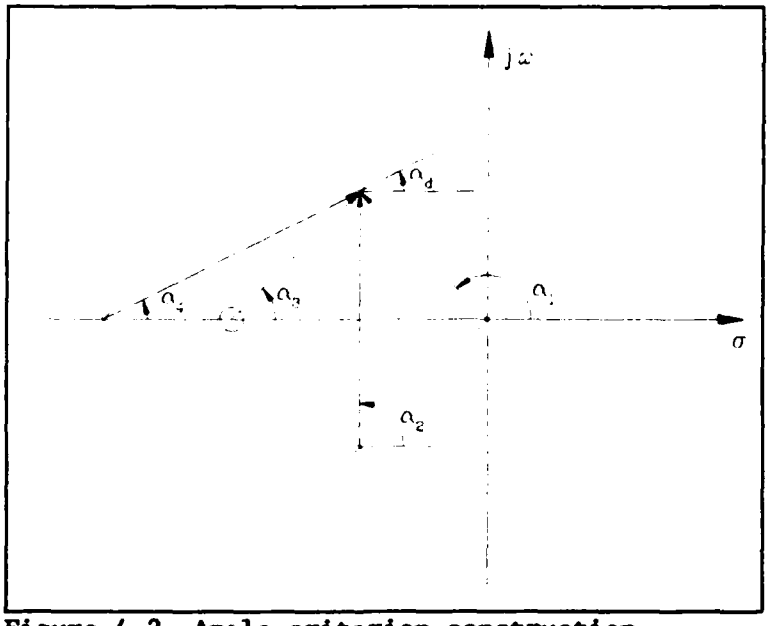

Figure 4.2 Angle criterion construction

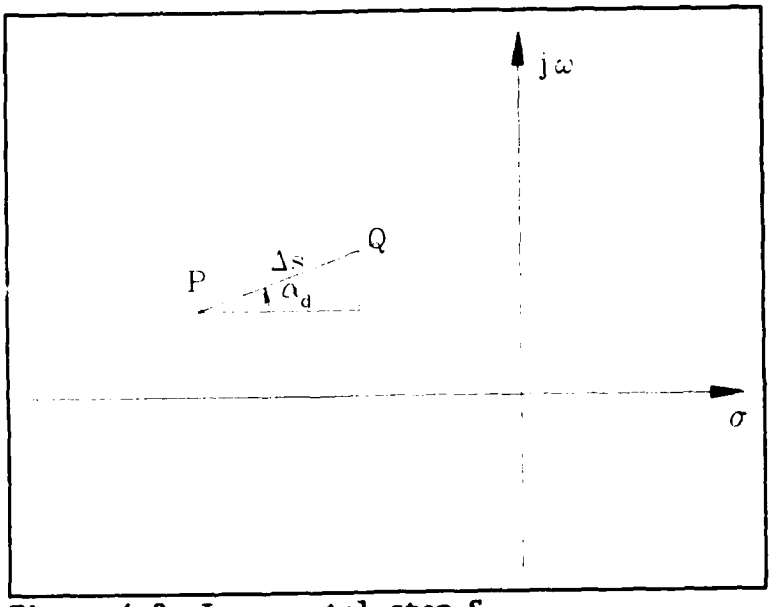

Figure 4-3 Incremental step S_Q

BLOCK 7 In order for a point $s=\sigma\pm j\omega$ to be on a root locus it must satisfy the equation

$$1 + KG(s) = 0 \tag{4.17}$$

or using (4.10)

$$1 + K(X+jY) = 0 \tag{4.18}$$

or

$$Y = 0 \tag{4.19}$$

$$X = -\frac{1}{K} \tag{4.20}$$

The approach to using equations (4.19) and (4.20) is to find values of $s=\sigma\pm j\omega$ that satisfy (4.19) and use these in (4.20) to calculate the corresponding value of K. Since it is unreasonable to calculate values that will make (4.19) exactly equal to zero, a small error value can be established to minimize Y.

$$|Y| \leq \text{error} \tag{4.21}$$

BLOCK 8 If the point S_Q is far enough away from the root locus that (4.21) is not true, a search at right angles to the line $(S_Q\text{-}S_P)$ is done to find a point S_R sufficiently close to satisfy (4.21). This is shown in figure 4.4.

$$S_R = S_Q + L_r(-\sin\alpha_d + j\cos\alpha_d) \tag{4.22}$$

In order to find the value for L_r, $G(s)=K[X+jY]$ is evaluated at two points along the line $(S_R\text{-}S_Q)$. The two points are determined by using (4.22) with two different values for L. They are

$$L_1=\Delta s 10^{-4}$$

$$L_2=2\Delta s 10^{-4}$$

These two trial points are applied to equations (4.5) to (4.9) and the resulting values $Y(L_1)$ and $Y(L_2)$ are interpolated together using (4.23) to minimize Y.

$$L_3 = L_2 - \frac{L_1 - L_2}{Y(L_1) - Y(L_2)} Y(L_1) \tag{4.23}$$

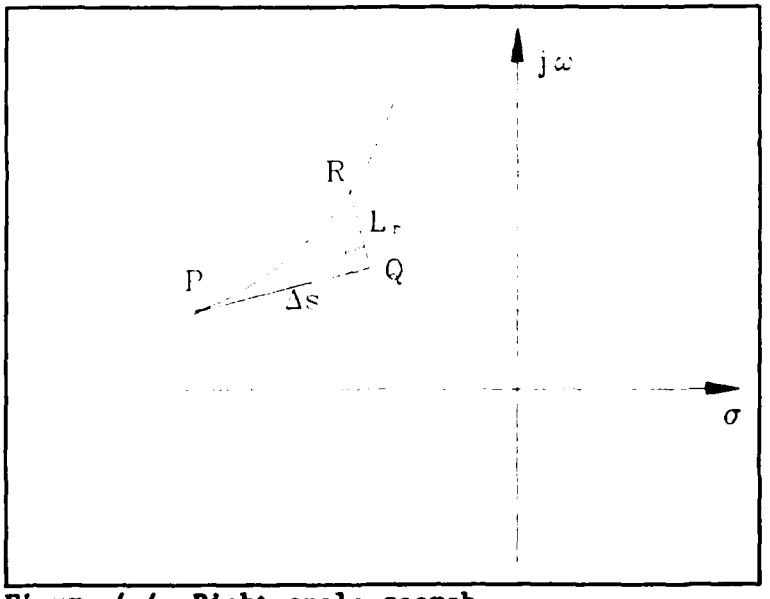

Figure 4.4 Right angle search

This process is repeated until $Y(L_r)$ satisfies (4.21) and at which point S_R is assumed to be found.

With the point S_R found it is possible to use equation (4.20) to find the corresponding gain, K. Furthermore, if the slope of the locus at S_R can be determined, the sequence can be repeated until the whole locus is found.

The slope at S_R can be found from:

$$m = \frac{d\omega}{d\sigma} = \frac{\sum_{k=1}^{N} \frac{\omega-\omega_{pk}}{r_{pk}^2} - \sum_{i=1}^{M} \frac{\omega-\omega_{zi}}{r_{zi}^2}}{\sum_{k=1}^{N} \frac{\sigma-\sigma_{pk}}{r_{pk}^2} - \sum_{i=1}^{M} \frac{\sigma-\sigma_{zi}}{r_{zi}^2}} \qquad (2.24)$$

where

$$r_{pk}^2 = (\sigma - \sigma_{pk})^2 + (\omega - \omega_{pk})^2$$

and

$$r_{zi}^2 = (\sigma - \sigma_{zi})^2 + (\omega - \omega_{zi})^2$$

Using the two points S_P and S_R, a new point $S_T=\sigma_T+j\omega_T$ can be calculated using the (usually correct) assumption that the slope of the curve is constant over a small interval on the locus.

$$\sigma_T = \sigma_S + \frac{1-m^2}{1+m^2}(\sigma_R-\sigma_P) + \frac{2m}{1+m^2}(\omega_R-\omega_P) \qquad (2.25)$$

$$\omega_T = \omega_S + \frac{2m}{1+m^2}(\sigma_R-\sigma_P) + \frac{1-m^2}{1+m^2}(\omega_R-\omega_P) \qquad (2.26)$$

Equations (2.25) and (2.26) are not used for values of $m < 0.01$ and $m > 100$, when horizontal and vertical slopes are assumed respectively.

BLOCKS 9, 10 Breakpoints in the locus are treated in this algorithm as the intersection of two loci, and if the wrong path is followed after such an intersection, the calculated value of K decreases, instead of

increasing. If a search step produces such a decrease, it is regarded as unsuccessful and a new step is taken at right angles to the unsuccessful one. Since breakpoints almost always occur at right angles, this strategy is usually successful.

BLOCK 11 A locus terminates either at a zero or at infinity on an asymptote. Infinity is assumed when any of the calculated coordinates exceeds the screen coordinates. An arrow is placed at the point of exit, pointing in the direction of the exit slope, to indicate infinity.

BLOCK 12 The locus is displayed as a set of data points joined together by straight line segments. Line segments lying on the left-hand plane, in the stable region, are displayed in green, while segments lying on the right-hand plane, in the unstable region, are displayed in red.

CHAPTER 5

BODE PLOTS

5.1 Introduction to Bode Plots

This chapter serves as an introduction to Bode plots which are used in the analysis and design of control systems. This is a graphical technique that plots a control system's gain amplitude and phase angle response curves against the input frequency. It is customary to plot the gain in decibels and the phase angle in degrees against the common logarithm of the input frequency. This is the form first introduced by H.W. Bode. This discussion is not intended to serve as a complete guide to Bode plots, but rather an overview to help the reader to better understand the computer techniques discussed in the following chapter. For further details refer to reference [3].

5.2 Bode Plots

Consider the open-loop transfer function of a single-input single-output system expressed in complex frequency domain,

$$G(s)\big|_{s=j\omega} = G(j\omega) = A(\omega)+jB(\omega) = |G(j\omega)|\ \angle G(j\omega) \qquad (5.1)$$

where $G(j\omega)$ is a complex variable, while $A(\omega)$ and $B(\omega)$ are real variables. To convert $G(j\omega)$ to polar form, that is in magnitude and phase form, the

following equations are used:

$$|G(j\omega)| = (A^2(\omega)+B^2(\omega))^{1/2} \tag{5.2}$$

$$\phi(\omega) = \angle G(j\omega) = \tan^{-1}(B(\omega)/A(\omega)) \tag{5.3}$$

A Bode plot of this system consists of the gain and phase curves, defined as $20 \log|G(j\omega)|$ and $\phi(\omega)$ respectively, versus the angular input frequency ω.

Example: Consider a system with the following transfer function

$$G(s)=1/(1+\tau s) \tag{5.4}$$

Gain: The logarithmic gain is

$$20 \log|G| = 20 \log(1/(1+(\omega\tau)^2)^{1/2})$$

$$=-10 \log(1+(\omega\tau)^2) \tag{5.5}$$

In order to plot the gain versus input frequency, use the following forms for different ranges of the frequency. For $\omega \ll 1/\tau$, is approximated by

$$20 \log|G|=-10 \log(1) = 0 \ db, \ \omega \ll 1/\tau \tag{5.6}$$

For high frequencies, $\omega \gg 1/\tau$, the gain (5.5) becomes

$$20 \log|G| = -20 \log(\omega\tau), \ \omega \gg 1/\tau \tag{5.7}$$

while at $\omega = 1/\tau$, the form is

$$20 \log|G| = -10 \log(2) = -3.01 \ db \tag{5.8}$$

It is clear from (5.6) that the gain is a constant 0 *db* for low

frequencies, then it drops to -3.01 *db* at the *corner* or *break frequency* $\omega_c = 1/\tau$ and finally for high frequencies, the plot drops linearly at a slope of -20 *db/decade*, that is at a frequency range $\omega_1 \leq \omega \leq \omega_2$ such that $\omega_2 = 10\omega_1$. Therefore, because of this linearity, it is very convenient to plot the phase 20 $\log|G|$ versus $\log \omega$. This Bode magnitude plot is shown in figure 5.1.

Phase: The phase angle for (5.4) is given by

$$\phi(\omega) = -\tan^{-1}\omega\tau \tag{5.9}$$

which has the following asymptotic and exact behaviors for a wide range of frequencies.

$$\phi(\omega) \to 0°, \quad \omega \ll 1/\tau \tag{5.10}$$

$$\phi(\omega) = -45°, \quad \omega = 1/\tau \tag{5.11}$$

$$\phi(\omega) \to -90°, \quad \omega \ll 1/\tau \tag{5.12}$$

The phase plot is shown in figure 5.2.

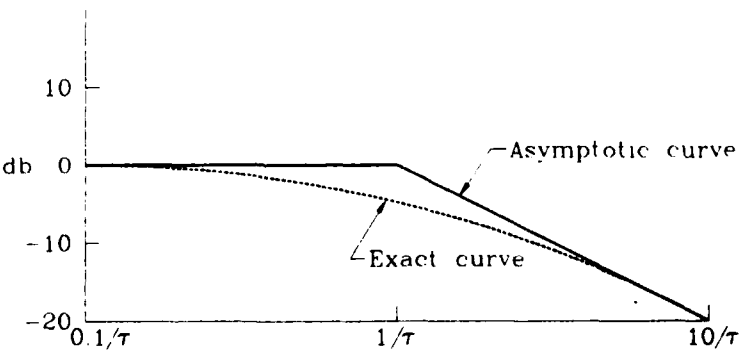

Figure 5.1 Bode magnitude plot

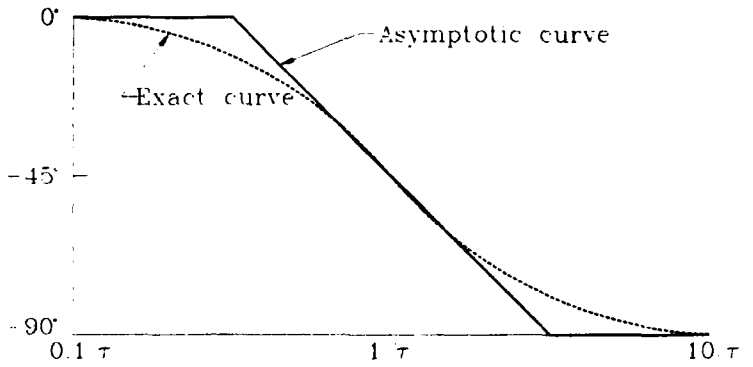

Figure 5.2 Bode phase plot

For rational transfer functions, it is only necessary to be able to plot gain and phase curves of the following types of terms:

1) Constant gain K.

2) Poles and zeros at the origin of the complex plane.

3) Real axis poles and zeros.

4) Complex conjugate pairs of poles and zeros.

A rational function can be factored into terms of above types, and the individual gain and phase curves plotted. The complete gain curve is the sum of the individual component gain curves, and the complete phase curve is the sum of the individual phase curves. A discussion of the four types of transfer function terms follows.

1.*Constant Gain K.* The logarithmic gain is 20 log K while the angle is 0°. The magnitude plots are horizontal lines. This is shown in figure 5.3.

2. *Poles or Zeros at the origin of the complex plane* $(j\omega)^{\pm 1}$. A pole or zero at the origin have a logarithmic gain as

$$20 \log|(j\omega)^{\pm 1}| = \pm 20 \log\omega \ db \qquad (5.13)$$

which is a straight line on a semi-log paper with slope of ±20 *db*/decade and a horizontal crossing at $\omega = 1$. The phase angle for this term is $\phi(\omega) = \pm 90^{\circ}$. Examples of poles and zeros at the origin can be seen in figures 5.4 and 5.5.

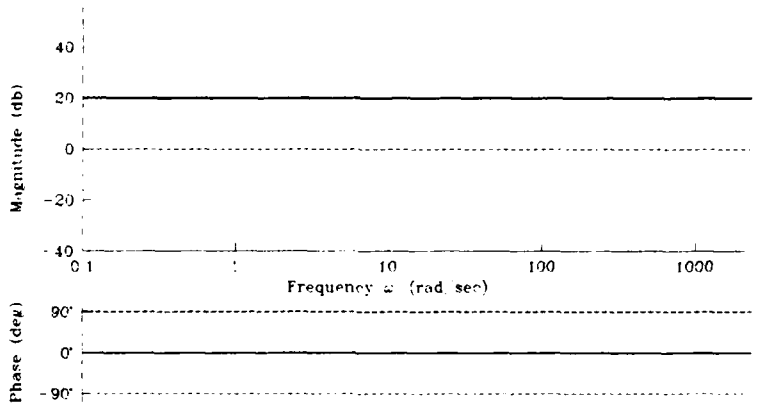

Figure 5.3 Bode plot with constant gain K=10

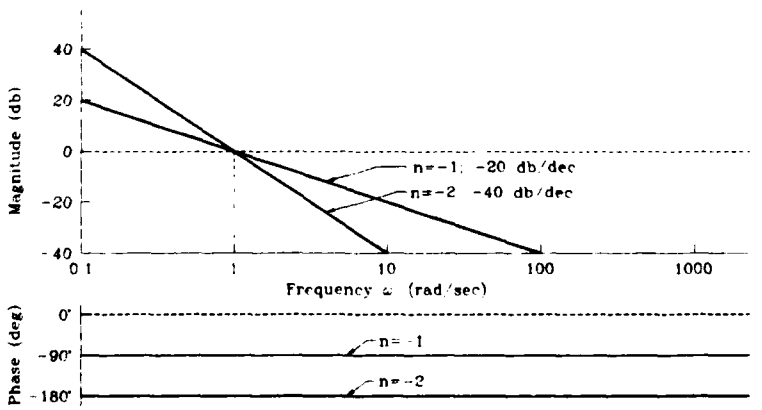

Figure 5.4 Bode plot with one and two poles at the origin

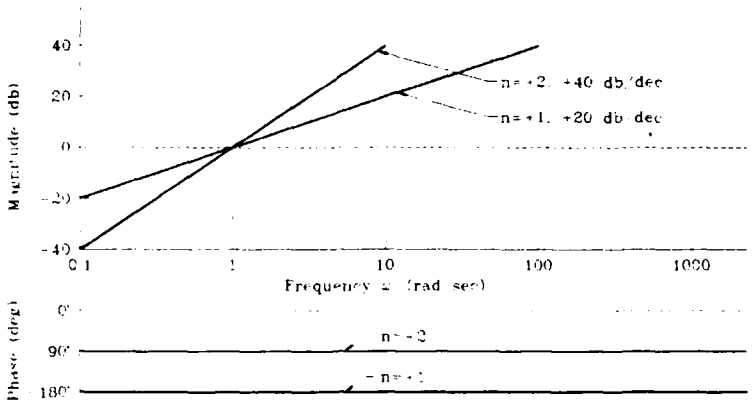

Figure 5.5 Bode plot with one and two zeros at the origin

---

3. *Poles or Zeros on Real Axis* $(1+j\omega/\omega_1)^{\pm 1}$. These terms have log-magnitude,

$$20 \log\left|(1+j\omega/\omega_1)^{\pm 1}\right| = \pm 10 \log(1+\omega/\omega_1)^2) \qquad (5.14)$$

The asymptotic behavior of this plot begins at $\pm 10 \log(\omega/\omega_1)^2 = \pm 20 \log(\omega/\omega_1)$ or a straight line with a ±20 *db*/decade slope. The two asymptotic lines (i.e. 0 db and ±20 db/dec.) cross each other at the point $\omega = \omega_1$ or at the corner frequency (see Fig. 5.1). However, the actual value of the logarithmic gain at $\omega = \omega_1$ is $\pm 10 \log(2) = \pm 3$ db as demonstrated with (5.4) example. The phase angle $\phi(\omega) = \pm \tan^{-1}(\omega/\omega_1)$ which begins at 0° for low frequencies ($\omega \ll \omega_1$), reaches $\phi(\omega_1) = \pm\tan^{-1}(1) = \pm 45^\circ$ and approaches $\phi(\omega) = \pm\tan^{-1}(\infty) = \pm 90^\circ$, (See Fig. 5.2). Examples can be seen in figures 5.6 and 5.7.

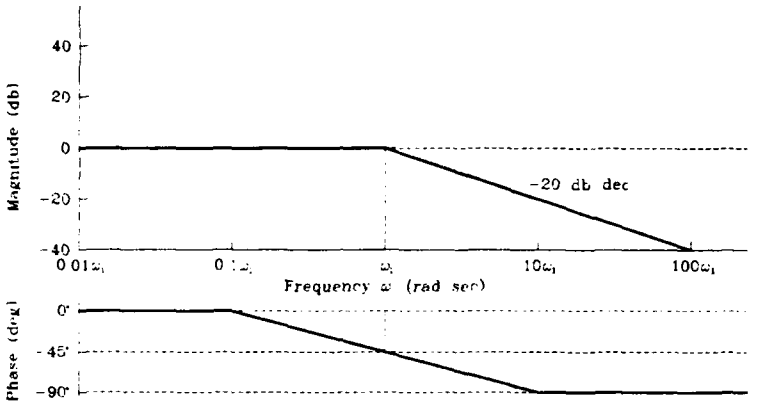

Figure 5.6 Bode plot with poles on the real axis

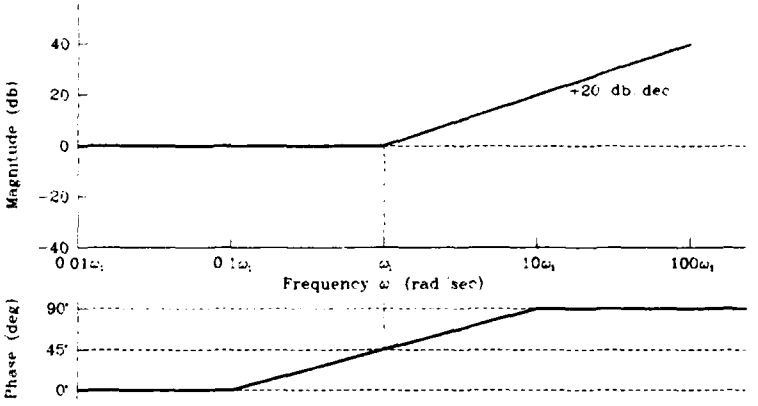

Figure 5.7 Bode plot with zeros on the real axis

---

4. *Complex Conjugate Poles or Zeros.* $[1+2\xi(j\omega/\omega_2)+(j\omega/\omega_2)^2]^{\pm 1}$. Let the ratio ω/ω_2 be represented by quantity ν and evaluate the logarithmic gain as,

$$\pm 20 \log|G(j\omega)| = \pm 10 \log((1-\nu^2)^2+4\xi^2\nu^2) \qquad (5.15)$$

while the phase angle is

$$\phi(\omega,\xi) = \tan^{-1}(2\xi\nu/(1-\nu^2)) \qquad (5.16)$$

Once again the asymptotic behaviors of the above plots will be investigated. When $\nu \ll 1$, the magnitude is,

$$\pm 10 \log(1) = 0 \ db \tag{5.17}$$

and the phase angle approaches 0°. On the other end of the frequency scale, i.e. for $\nu \gg 1$, the magnitude is

$$\pm 10 \log(\nu^4) = \pm 40 \log(\nu) \tag{5.18}$$

which results in a straight line with a slope of ±40 db/decade. The phase angle $\phi(\omega)$ approaches 0° asymptotically as $\nu \ll 1$ and reaches $\pm 180^\circ$ as $\nu \gg 1$.

The frequency ω_r at which the maximum magnitude occurs is called the *resonant frequency*. Note that as the damping ratio ξ approaches zero, the resonant frequency ω_r approaches the corner frequency ω_2. The resonant frequency is obtained by taking the derivative of the magnitude of (5.15) with respect to ν and setting it equal to zero. The resulting equation is

$$\nu^2 - 1 + 2\xi^2 = 0 \tag{5.19}$$

or

$$\omega_r = (1-2\xi^2)^{1/2}, \ \xi < 0.707 \tag{5.20}$$

The maximum value of the magnitude itself is

$$M_r = |G(\omega_r)| = (2\xi(1-\xi^2)^{-1}, \ \xi < 0.707 \tag{5.21}$$

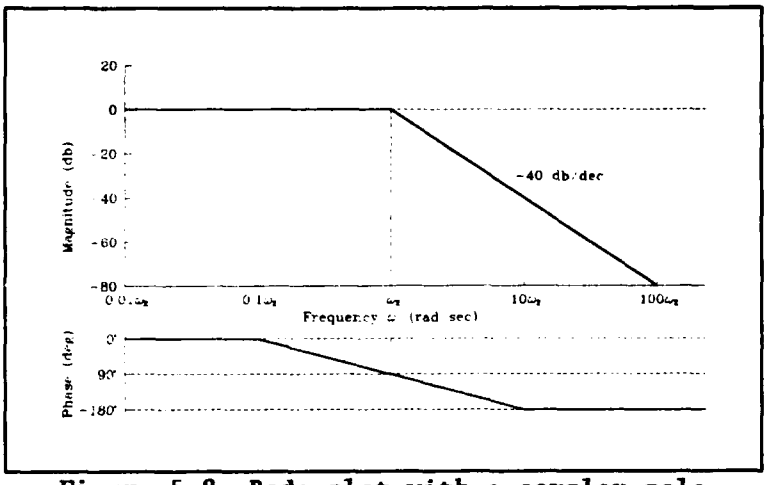

Figure 5.8 Bode plot with a complex pole

CHAPTER 6

COMPUTER-AIDED BODE PLOTS

6.1 Introduction to Computer-Aided Bode Plots

This chapter describes the Bode plot algorithm as shown in the computer flow chart in Figure 6.1. In the interest of simplicity, some of the less significant steps are not shown. For more detail, refer to the program listing in Appendix D.

6.2 Bode Program Description

BLOCK 1 Input the open-loop transfer function information, the number of poles and zeros and their corresponding complex roots.

BLOCK 2 Input the open-loop transfer function gain K and the lower and upper plotting frequency range values, omegal $\ll \omega \ll$ omegau. If the lower frequency value omegal is not a power of 10, i.e. not 0.001, 0.01..., then it's converted to the next lower power of 10. Likewise, the upper frequency is converted to the next higher power of ten. As an example, if the frequency range values were input as omegal=0.05 and omegau=300, then their corresponding converted values would be initial=0.001 and final=1000. This conversion is done in order that the abscissa on the output plot has a starting and ending points with powers of 10.

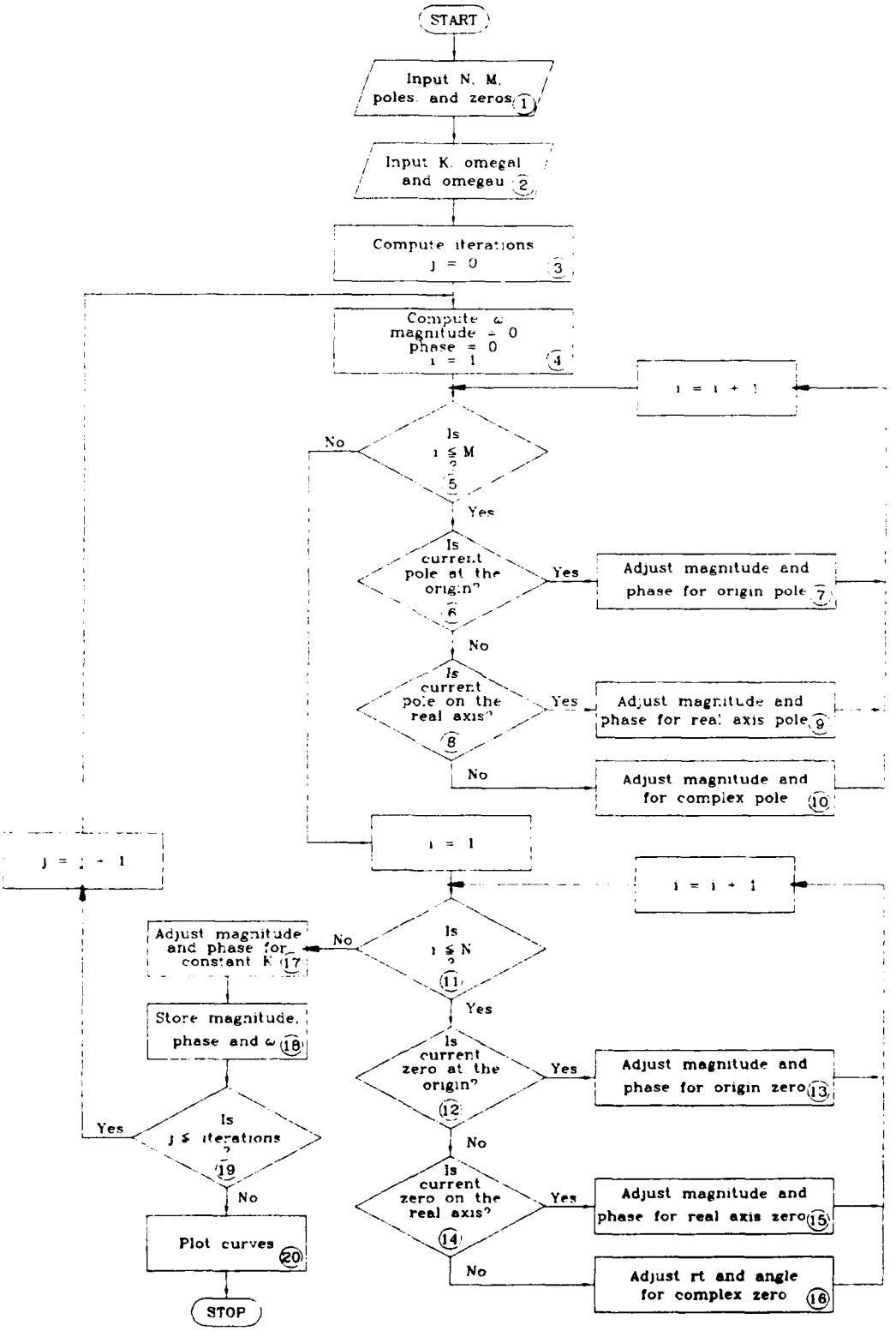

Figure 6.1 Bode plot flow chart

BLOCK 3 Compute the number of frequency iterations. This is a function of the frequency range and the number of available horizontal output screen pixels. This insures that the minimum number of iterations are used that will produce the most accurate plot with the least amount of computer run time. Next, set the frequency loop counter, j, equal to zero.

BLOCK 4 For each iteration of the frequency loop a new frequency value, ω, must be computed. ω is given by Equation 6.1

$$\omega = \text{omegal} + j\Delta\omega \tag{6.1}$$

where $\Delta\omega$ is the frequency step size.

Also, for each loop pass through the frequency loop, the running totals of the magnitude and phase and the pole loop counter, i, are set equal to zero.

BLOCK 5 For each iteration of the frequency loop, each of the M factored poles are evaluated to adjust the magnitude and phase running totals. Block 5 controls the poles loop and sends the program flow to the zero loop once each of the poles are evaluated.

As described in chapter 5, the transfer function can be factored into four types of terms, they are

1) Constant gain K.

2) Poles and zeros at the origin of the complex plane.

3) Real axis poles and zeros.

4) Complex poles and zeros.

BLOCK 6 Checks for poles at the origin and if true sends the flow to block 7.

BLOCK 7 *Pole at the origin.* Adjusts the magnitude and phase according to the following equations.

$$\text{magnitude} = \text{magnitude} - 20 \log_{10}(\omega) \tag{6.2}$$

$$\text{phase} = \text{phase} - \pi/2 \tag{6.3}$$

BLOCK 8 Checks for real axis poles and if true sends flow to block 9.

BLOCK 9 *Real axis pole.* Adjusts the magnitude and phase according to the following equations.

$$\text{magnitude} = \text{magnitude} - 20 \log_{10}(1+\omega/\omega_p) \tag{6.4}$$

$$\text{phase} = \text{phase} - \tan^{-1}(\omega/\omega_p) \tag{6.5}$$

BLOCK 10 *Complex pole.* If blocks 6 and 8 are both false the flow comes to block 10 and the magnitude and phase are adjusted as follows.

$$\text{magnitude} = \text{magnitude} - 20 \log_{10}((1-(\omega/\omega_p)^2)^2 + 4\xi^2(\omega/\omega_p)^2)^{1/2} \tag{6.6}$$

$$\text{phase} = \text{phase} - \tan^{-1}(2\xi(\omega/\omega_p)/(1-(\omega/\omega_p)^2)) \tag{6.7}$$

Once all of the poles are evaluated, the loop counter i is set equal to 1 and the flow goes to the zero loop control block 11.

BLOCK 11 For each iteration of the frequency loop each of the N factored zeros are evaluated to adjust the magnitude and phase running totals. Block 11 controls the zero loop and sends the program flow to the constant K block once each of the zeros are evaluated.

BLOCK 12 Checks for zeros at the origin and if true sends the flow to block 13.

BLOCK 13 *Zero at the origin.* Adjusts the magnitude and phase according to the following equations.

$$\text{magnitude} = \text{magnitude} + 20 \log_{10}(\omega) \tag{6.8}$$

$$\text{phase} = \text{phase} + \pi/2 \tag{6.9}$$

BLOCK 14 Checks for real axis zeros and if true sends flow to block 15.

BLOCK 15 *Real axis zero.* Adjusts the magnitude and phase according to the following equations.

$$\text{magnitude} = \text{magnitude} + 20 \log_{10}(1+\omega/\omega_p) \tag{6.10}$$

$$\text{phase} = \text{phase} + \tan^{-1}(\omega/\omega_p) \tag{6.11}$$

BLOCK 16 *Complex zero.* If blocks 12 and 14 are both false the flow comes to block 16 and the magnitude and phase are adjusted as follows.

$$\text{magnitude} = \text{magnitude} + 20 \log_{10}((1-(\omega/\omega_p)^2)^2 + 4\xi^2(\omega/\omega_p)^2)^{1/2} \tag{6.12}$$

$$\text{phase} = \text{phase} + \tan^{-1}(2\xi(\omega/\omega_p)/(1-(\omega/\omega_p)^2)) \qquad (6.13)$$

Once all of the poles are evaluated, the flow goes to the constant control block 17.

BLOCK 17 Once all of the poles and zeros of the transfer function are evaluated the magnitude is adjusted for the constant gain term K. Only the magnitude is adjusted since the phase change for a constant is zero.

$$\text{magnitude} = \text{magnitude} + 20\ \log_{10}(K) \qquad (6.14)$$

BLOCK 18 The current frequency, magnitude and phase data is stored once all of the terms of the transfer function are evaluated.

BLOCK 19 This block controls the frequency iteration loop and sends flow to the output block once the transfer function is evaluated for each of the step frequencies. If not finished, the iteration loop counter is increase by one and the loop continues.

BLOCK 20 Flow is sent here once all of the data points that describe the magnitude and phase curves are computed. Both curves are plotted together sharing the same frequency abscissa. The left-hand vertical scale represents the magnitude as measured in decibels and is shown in magenta. The right-hand vertical scale measures the phase angle in degrees and is shown in green. The magnitude curve is plotted with a solid magenta curve and the phase is plotted as a dotted green curve. The above color information only applies to plots run on a machine with an IBM compatible Enhanced Graphics Adapter (EGA). Plots run on a machine with an IBM

compatible Color Graphics Adapter (CGA) are only shown in black and white due to the low resolution of the CGA when plotting in color. The magnitude is plotted with a solid white curve while the phase is plotted with a dotted white curve.

Hard copy prints of the plots are provided by way of the print screen function. If a printout is desired, press the P key and the output is automatically sent to the printer. A example output screen is shown in figure 6.2.

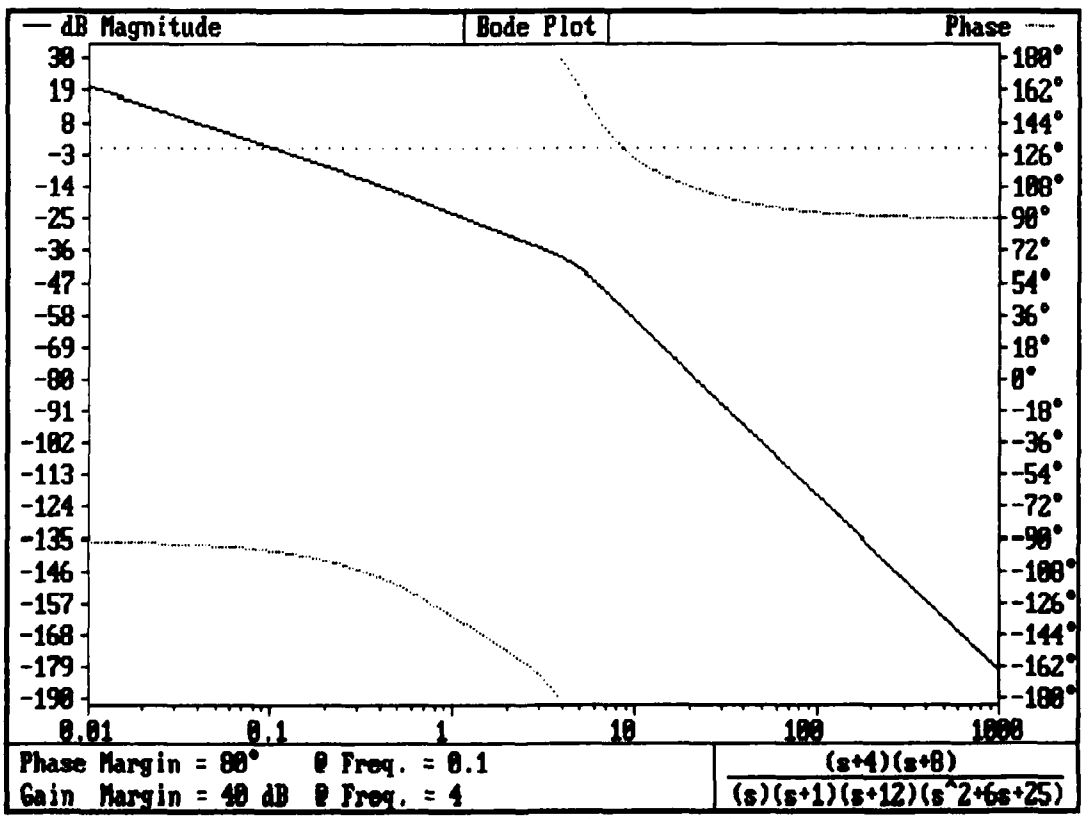

Figure 6.2 Example Bode plot

CHAPTER 7

SUMMARY

7.1 Thesis Summary

With the wide spread availability of the personal computers to the control systems engineer, is would only make sense to supplement the time-honored techniques of using root locus and bode plots in the analysis and design of control systems with one that involves computer-aided support. Using the computer to do the mundane task of generating the graphs allows the designer to spend more time analyzing the system. This could also allow a student in a controls course the opportunity to learn more of the theory behind control systems analysis instead of spending all their time learning the tedious process of generating the graphs by hand. It was the purpose of this thesis to develope computer algorithms to automate this task in a easy to use format. This software is called the *Control Systems Software Package* (CSSP).

REFERENCE

1. Hale, Francis J.: Introduction to Control System Analysis and Design. Prentice-Hall Inc., NJ, 1973.

2. Borrie, John A.: Modern Control Systems: A Manual of Design Methods. Prentice-Hall International (UK) Ltd, 1986.

3. Jamshidi, M. and Malek-Zavarei, M.: Linear Control Systems: A Computer-Aided Approach. First Edition. Pergamon Books Ltd., NY, 1986.

4. McDonald, A.C. and Lowe, H.: Feedback and Control Systems. Reston Publishing Company, Inc., Reston, Virginia, 1981.

5. Atkinson, P. and Dalvi, V.S.: An Improved Algorithm for the Automatic Determination of Roots Loci, Radio and Electronic Engineer, 41, 1971, 365.

6. Hostetter, G.H. , Savant, C.J. Jr. and Stefani, R.T.: Design of Feedback Control Systems. CBS College Publishing, NY, 1982.

7. Gerald, C.F. and Wheatley, P.O.: Applied Numerical Analysis. Third Edition. Addison-Wesley Publishing Company, Reading Massachusetts, 1985.

APPENDIX A

CSSP MAKE SOURCE CODE

```
############################  START OF MAKE FILE ############################
#                                                                           #
#  This is the make description file used in the development of CSSP.  This #
#  make file was used with the Microsoft Make utility.  In CSSP development, #
#  the Make utility automatically updated the executable file whenever any  #
#  of the source or object files were altered.                              #
#                                                                           #
#   To use the Make utility and this file, type at the DOS prompt           #
#                                                                           #
#   MAKE FILENAME                                                           #
#                                                                           #
#   press ENTER.                                                            #
#                                                                           #
#############################################################################

cssp.obj:       cssp.c
     cl /AL /DMSCV4 cssp.c /c

locus.obj:    locus.c
     cl /AL /DMSCV4 locus.c /c

bode.obj:       bode.c
     cl /AL /DMSCV4 bode.c /c

modified.obj:      modified.c
     cl /AL /DMSCV4 modified.c /c

factored.obj:   factored.c
     cl /AL /DMSCV4 factored.c /c

coeff.obj:    coeff.c
     cl /AL /DMSCV4 coeff.c /c

video.obj:    video.c
     cl /AL /DMSCV4 video.c /c

root.obj:       root.c
     cl /AL /DMSCV4 root.c /c

printer.obj:    printer.c
     cl /AL /DMSCV4 printer.c /c

cssp.exe:   cssp.obj locus.obj bode.obj modified.obj factored.obj\
          coeff.obj video.obj root.obj printer.obj
#
     link /NOE /ST:10000 cssp+locus+bode+modified+factored+coeff+\
     video+root+printer,,,lwin+llibce;

############################  END OF MAKE FILE ############################
```

APPENDIX B

CONTROL SYSTEMS SOFTWARE PACKAGE SORCE CODE

```
/************************  START OF CSSP ROUTINE  ************************/
/*                                                                       */
/*    This is the main program driver for CSSP.  Control to each of the  */
/*    seperate routines, such as the Bode plot routine, starts from and  */
/*    and returns to this program.                                       */
/*                                                                       */
/*    This files also includes the source for the following funcions:    */
/*                                                                       */
/*    void tranfunc():     Transfer function calling routine             */
/*    sign(double):   Check sign of number                               */
/*                                                                       */
/************************  START GLOBAL VARIABLES  ************************/
#include        "variables.h"                       /* Include variable list */

  struct mitem {                                      /* menu item template */
    int r;                                                      /* row */
    int c;                                                      /* col */
    char *t;                                                   /* text */
    int rv;                                              /* return value */
  };

  struct pmenu {                                    /* popup menu structure */
    WINDOWPTR wpsave;                          /* a place for the window handle */
    int winopn;                                     /* window open flag */
    int lndx;                                              /* last index */
    int fm;                                        /* first menu item index */
    int lm;                                         /* last menu item index */
    struct mitem scrn[25];                         /* a bunch of menu items */
  };

int     sign(double );                                /* Check sign of number */
/************************  END GLOBAL VARIABLES  ************************/
/************************  START OF MAIN CSSP ROUTINE  ************************/
main()
   {
   WINDOWPTR   w1, w2;                                      /* a few windows */
   WINDOWPTR   qpopup();                                 /* function returns WP */
   int i;                                                  /* scratch integers */
   int watrib,batrib;                                     /* scratch atributes */
   int rv;                                                        /* for popup */
   int tranfunc();                                  /* Transfer function routine */

    static struct pmenu m1 = {                                    /* Main menu */
       00, FALSE, 00,
       06,  9, {
       01, 02, "                          Main Menu", 0,
       02, 02, "", 0,
       03, 00, "──────────────────────────────────────────────────────────────",0,
       04, 00, "         Press the desired menu number or position the", 0,
       05, 00, "       highlight bar with the cursor keys and press enter.", 0,
       06, 00, "──────────────────────────────────────────────────────────────",0,
      8, 13, "[1] Input Transfer Function Information", 1,
       10, 13, "[2] Root Locus Plot", 2,
       12, 13, "[3] Bode Plot", 3,
       14, 13, "[4] Quit CSSP and go to DOS", 4,
       99, 99, "",99 }
       };


   set_print_screen();                             /* Load print screen program */

   _setvideomode(_TEXTC80);                          /* Set video mode to text */

   wn_dmode(FLASH);                                 /* Set window speed to fast */

    for(;;)
       {
       /*
       * Set window attributes:
       *
       *       border - blue/white box
       *       window - white background/black letters
```

```
        *
        */
        batrib = v_setatr(BLUE,WHITE,0,0);
        watrib = v_setatr(BLUE,WHITE,0,0);

        /*
        * Open title window
        */
        w1 = wn_open(0,0,0,78,23,watrib,batrib);
        if(!w1) exit(1);
        wn_titla(w1," Control Systems Software Package ",batrib);

        /*
        * Open credit window at bottom middle row
        */
        w2 = wn_open(1000,24,19,39,1,watrib,batrib);
        if(!w2) exit(1);
        wn_printf(w2," Copyright (c) 1988-1989 Carl F. Adams ");


        rv = popup(0,3,6,65,16, WHITE<<4|BLACK, BLUE<<4|WHITE, &m1,TRUE);

        switch (rv)
        {
        case 1:
           transfunc();                        /* Transfer function routine */
           wn_close(w2);
           wn_close(w1);
           break;
        case 2:
           wn_close(w2);
           wn_close(w1);
           output_screen();                    /* Initialize output screen */
           plot(m,n);                           /* Root locus plot routine */
           printer();
           _setvideomode(_TEXTC80);             /* Set video mode to text */
           break;
        case 3:
           bode(m,n);                                /* Bode plot routine */
           printer();
           _setvideomode(_TEXTC80);             /* Set video mode to text */
           break;
        case 4:
           wn_close(w1);
           wn_close(w2);
           exit(0);
        }

       }
}
/************************  END OF MAIN CSSP ROUTINE  ************************/

/*********************  START TRANSFER FUNCTION ROUTINE  *********************/
/*                                                                           */
/*   This routine prompts the user for what type of transfer function        */
/*   is to be plotted.  A popup menu of the form types is displayed and      */
/*   control is sent to the designated function once its' function menu      */
/*   number is pressed or else the function is highlighted with the          */
/*   highlight bar and the ENTER key is pressed.                             */
/*                                                                           */
/*****************************************************************************/
int transfunc()
{
   int rv;                                                  /* for popup */

   static struct pmenu m2 = {
      00, FALSE, 00,
      14, 16, {
      00, 03, "                    Transfer Function Forms",0,
      01, 03, "", 0,
      02, 03, "                                                    ",0,
      03, 03, " Factored Form      G(s) = K(s + C1)(s + C2)(s^2 + C3s + C4) ...  ",0,
```

```
      04, 03, "                              (s + C5)(s + C6)(s^2 + C7s + C8) ...  ",0,
      05, 03, "", 0,
      06, 03, "                            K{C1s^m + C2s^(m-1) + ... + C3s + C4}  ",0,
      07, 03, " Polynomial Form    G(s) = ------------------------------------------  ",0,
       8, 03, "                             {C5s^n + C6s^(n-1) + ... + C7s + C8}  ",0,
       9, 03, "", 0,
      10, 00,
"_____________________________________________________________________________",0,
      11, 03, "     Press the desired function form number or position the", 0,
      12, 03, "       highlight bar with the cursor keys and press enter.", 0,
      13, 00,
"_____________________________________________________________________________",0,
      15, 27, "[1] Factored Form", 1,
      17, 27, "[2] Polynomial Form", 2,
      19, 27, "[3] Return to Main Menu", 3,
      99, 99, "",99 }
      };
   rv = popup(0,1,3,72,21, WHITE<<4|BLACK, BLUE<<4|WHITE, &m2,TRUE);

   switch (rv)
      {
      case 1:
       rv=factored();                              /* Factored polynomials */
       return(1);
      case 2:
       coeff();                                  /* Non-factored polynomials */
       rv=  ROOT(&pcoef[0],&poles[0][0],n);        /* Find the pole roots */
       rv=  ROOT(&zcoef[0],&zeros[0][0],m);        /* Find the zero roots */
       return(2);
      case 3:
       return(3);                                  /* Return to Main Menu */
      }
}
/*********************** END TRANSFER FUNCTION ROUTINE ********************/

/************************** START SIGN ROUTINE ******************************/
/*                                                                          */
/*      This routine checks for the sign of the variable x.  It returns a   */
/*      value of 0 for small values of x, or it returns a value of 1 if x   */
/*      is equall to 0, and if the above is not true it returns the sign    */
/*      of x.                                                               */
/****************************************************************************/
int     sign(double x)
{
   if(fabs(x)<0.00000000001 && x!=0.0)
      return(0);
   else if(x==0.0)
      return(1);
   else
      return(x/fabs(x));
}
/************************** END OF SIGN ROUTINE ****************************/

/***************************** END OF CSSP FILE ****************************/
```

APPENDIX C

ROOT LOCUS SOURCE CODE

```
/************************  START OF LOCUS ROUTINE  ************************/
/*                                                                        */
/*    This is program plots the root locus of a given transfer function.  */
/*                                                                        */
/************************  START GLOBAL VARIABLES  ************************/

#include        "variables.h"                       /* Include variable list */

double  sigmap[10],                                    /* Real part of pole */
      sigmaz[10],                                    /* Real part of zero */
      omegap[10],                                 /* Imaginary part of pole */
      omegaz[10],                                 /* Imaginary part of zero */
      Xp,Yp;                              /* X & Y components of present point */

/************************  END GLOBAL VARIABLES  ************************/
/******************* START OF MAIN LOCUS PLOT ROUTINE ********************/
void    plot(int n,int m)
{

      int     i,j,k,l,                                      /* Loop counters */
              locus,                                        /* Branch counter */
              converge_ck,              /* Convergence check 1=True 0=False */
              repeat,                                       /* Repeated roots */
              repeat_counter=1,                   /* Repeated roots counter */
              breakout,                    /* Breakout check 1=True 0=False */
              infinite_red=0,           /* Check for red branch going to ∞ */
              infinite_green=0,         /* Check for green branch going to ∞ */
              green_length=0,             /* Length of green output string */
              red_length=0;                 /* Length of red output string */

      double  theta,                                 /* Present branch angle */
              range,             /* Allowable range for convergence to zero */
              X_max,                          /* Maximum allowable X value */
              Y_max,                          /* Maximum allowable Y value */
              Y_min,                          /* Minimum allowable Y value */
              alpha[9],                            /* Array of pole angles */
              beta[9],                             /* Array of zero angles */
              pi=3.141592654,
              X1,X2,X3,                             /* Sequential X values */
              Y1,Y2,Y3,                             /* Sequential Y values */
              midx,midy,                            /* Middle index values */
              GREEN_max=0.0,                 /* Maximum green branch value */
              GREEN_min=0.0,                 /* Minimum green branch value */
              RED_max=0.0,                     /* Maximum red branch value */
              RED_min=0.0,                     /* Minimum red branch value */
              x,                                   /* Present branch slope */
              K,                                     /* Present gain value */
              K1,K2,K3,                           /* Sequential gain values */
              Yp1,Yp2,Yp3,                     /* Sequential deviation values */
              L1,L2,L3,                        /* Sequential length modifiers */
              error=0.00001,         /* Maximum allowable branch deveiation */
              XX,YY;                           /* Temporary position values */


      char    green_string[20],                  /* Red branch string buffer */
              red_string[20];                  /* Green branch string buffer */

      double  slope(int,int,double,double);                 /* Slope routine */
      double  Y(int ,int ,double ,double );  /* Position equation routine */
      double  gain(int ,int ,double ,double );               /* Gain routine */
      int     converge(int , double, double, double );      /* Converged ? */
      void    sort(int );         /* sort array values (lowest -> highest) */


  delta_s=160.0/max/75;
  range=delta_s;

  L1=delta_s*0.0001;
  L2=delta_s*0.0002;
```

```
X_max=(vc.numxpixels-4)/2/max;
Y_max=X_max*3/4;
Y_min=-(Y_max-32/max*4/3);

sort(m);

for(i=1;i<=m;i++)                    /* Load real & imaginary pole values */
   {
   sigmap[i]=poles[i-1][1];
   if(fabs(poles[i-1][2])<error)
    omegap[i]=0;
   else
    omegap[i]=poles[i-1][2];
   }

for(i=1;i<=n;i++)                    /* Load real & imaginary zero values */
   {
   sigmaz[i]=zeros[i-1][1];
   if(fabs(zeros[i-1][2])<error)
    omegaz[i]=0;
   else
    omegaz[i]=zeros[i-1][2];
   }

for(locus=1;locus<=m;locus++)               /* Start of main branch loop */
   {
   k=0;
BEGIN:  X1=sigmap[locus];
   Y1=omegap[locus];
   _moveto(X1*max,-Y1*aspect_ratio*max);

   theta=0.0;
   repeat=0;
   for(i=1;i<=m;i++)           /* Compute pole departure angle component */
    {
    x=sign(sigmap[i]);
    if( fabs(X1-sigmap[i])<=range && fabs(Y1-omegap[i])<=range )
       {
       repeat++;
       alpha[i]=0.0;
       }
    else
       alpha[i]=atan2((Y1-omegap[i]),(X1-sigmap[i]));

    theta=theta-alpha[i];
    }
   for(i=1;i<=n;i++)           /* Compute zero departure angle component */
    {
    if(X1==sigmaz[i]&&Y1==omegaz[i])
       beta[i]=0.0;
    else
       {
       beta[i]=atan2((Y1-omegaz[i]),(X1-sigmaz[i]));
       }
    theta=theta+beta[i];
    }

   if(repeat>1)                             /* Check for repeated roots */
    {
    theta=(theta+pi)/repeat + 2*pi*(repeat_counter++-1)/repeat;
    if(repeat_counter>repeat)
       repeat_counter=1;
    }
   else
    theta=theta-pi;

   X2=X1+1.5*delta_s*cos(theta);
   Y2=Y1+1.5*delta_s*sin(theta);

   Yp2=Y(m,n,X2,Y2);                    /* Check for branch convergence */
   while(fabs(Yp2)>error)               /* and itterate as needed.      */
    {
```

```
            XX=X2;
            YY=Y2;
            X2=XX-L1*sin(theta);
            Y2=YY+L1*cos(theta);
            Yp1=Y(m,n,X2,Y2);

            X2=XX-L2*sin(theta);
            Y2=YY+L2*cos(theta);
            Yp2=Y(m,n,X2,Y2);

            L3=L1-(L1-L2)/(Yp1-Yp2)*Yp1;
            if(fabs(L3)>delta_s/4.0)
               L3=sign(L3)*delta_s/4.0;

            X2=XX-L3*sin(theta);
            Y2=YY+L3*cos(theta);
            Yp2=Y(m,n,X2,Y2);
            theta=atan((Y2-Y1)/(X2-X1));
            }

           if(X2>0.0)                          /* Plot in red if branch is unstable */
            _setcolor(RED);
           else                               /* else it is stable and plot in green */
            _setcolor(GREEN);
           _lineto(X2*max,-Y2*aspect_ratio*max);

           K2=-1.0/Xp;

           if(fabs(Y2)>delta_s/2)
            breakout=1;
           else
            breakout=0;

           l=j=0;
           while(j<=800)                          /* Compute and plot locus branch */
            {
START:      j++;
            x=slope(n,m,X2,Y2);

            if(fabs(x)<0.01)                       /* Check for horizontal slope */
               {
               X3 = X2 + sign(X2-X1)*delta_s*2;
               Y3=Y2;
               theta=sign(X2-X1)*pi;
               }
            else
               if(fabs(x)>100.0&&fabs(Y2)>delta_s)   /*Check for vertical slope*/
                  {
                  X3=X2;
                  Y3 = Y2 + sign(Y2-Y1)*delta_s*2;
                  theta=sign(Y2-Y1)*pi*0.5;
                  }
            else
               if(fabs(x)>100.0&&j>1)
                  {
                  X3=X2;
                  Y3 = Y2 - sign(x)*delta_s*2;
                  theta=sign(Y2-Y1)*pi*0.5;
                  }
            else
               {
               X3 = X2 + (1.0-x*x)/(1.0+x*x)*sign(X2-X1)*delta_s +
                  (2.0*x)/(1.0+x*x)*sign(Y2-Y1)*delta_s;
               Y3 = Y2 + (2.0*x)/(1.0+x*x)*sign(X2-X1)*delta_s +
                  (1.0+x*x)/(1.0+x*x)*sign(Y2-Y1)*delta_s;
               theta=atan((Y3-Y2)/(X3-X2));
               }
                   /* Check for break-in point */
            if(sign(Y3)!=sign(Y1)&&fabs(Y3)>error&&fabs(Y1)>error&&breakout==1)
               {
               X2=X2-Y2*(X3-X2)/(Y3-Y2);
               X3=X2+sign(Y3)*delta_s*2;
```

```
            Y2=Y3=sign(Y1-Y2)*error;
            Yp2=Y(m,n,X2,Y2);
            K2=-1.0/Xp;
            }
Y3:         Yp3=Y(m,n,X3,Y3);          /* Again, check for branch convergence */
          while(fabs(Yp3)>error)       /* and adjust as needed.               */
            {
            XX=X3;
            YY=Y3;
            X3=XX-L1*sin(theta);
            Y3=YY+L1*cos(theta);
            Yp1=Y(m,n,X3,Y3);

            X3=XX-L2*sin(theta);
            Y3=YY+L2*cos(theta);
            Yp2=Y(m,n,X3,Y3);
            L3=L1-(L1-L2)/(Yp1-Yp2)*Yp1;
            if(fabs(L3)>delta_s/4.0)
               L3=sign(L3)*delta_s/4.0;

            X3=XX-L3*sin(theta);
            Y3=YY+L3*cos(theta);
            Yp3=Y(m,n,X3,Y3);
            j++;
            if(j>800) goto NEXT;              /* Check for non-convergence, */
            }                                 /* is it going no where?      */

          K3=-1.0/Xp;

          if(l++>10)
            {
            l=0;
            midx=X3;
            midy=Y3;
            }

          if(K3<K2)                        /* Check to make sure the branch is */
            {                              /* going in the correct direction.  */
            converge_ck=converge(n,range,X2,Y2);
            if(converge_ck > 0)
               goto NEXT;

            Yp1=Y(m,n,X1,Y1);
            Yp2=Y(m,n,X2,Y2);
            Yp3=Y(m,n,X3,Y3);

            if(sign(Yp1)!=sign(Yp2))
               {
               X2=X2-(X2-X1)/(Yp2-Yp1)*Yp2;
               X1=X2;
               Y2=Y3=sign(Y1-Y2)*error;
               breakout=1;
               }
            else
               {
               X2=X2-(X3-X2)/(Yp3-Yp2)*Yp2;
               X1=X2;
               Y2=Y3=sign(Y1-Y2)*error;
               breakout=1;
               }

CHECK:      Yp2=Y(m,n,X2,Y2);
            while(fabs(Yp2)>error)
               {
               XX=X2;
               YY=Y2;
               X2=XX+L1;
               Yp1=Y(m,n,X2,Y2);

               X2=XX-L1;
               Yp3=Y(m,n,X2,Y2);
               L3=L1*Yp1/(Yp2-Yp1);
```

```
            if(fabs(L3)>delta_s/4.0)
               L3=sign(L3)*delta_s/4.0;

            X2=XX+L3;
            Yp2=Y(m,n,X2,Y2);
            theta=atan((Y2-YY)/(X2-XX));
            j++;
            if(j>800) goto NEXT;
            }
         K2=-1.0/Xp;
         Y2=Y2-sign(X3-X1)*delta_s;
         goto START;
         }
      else
         {
         if(X3>0.0)
            {
            forcolor2=RED;
            _setcolor(forcolor2);
            if(RED_max==0.0 && RED_min==0.0 && j!=1)
               RED_min=K2-X2*(K3-K2)/(X3-X2);
            if(K3>RED_max) RED_max=K3;
            if(K3<RED_min) RED_min=K3;
            }
         else
            {
            forcolor2=GREEN;
            _setcolor(forcolor2);
            if(GREEN_max==0.0 && GREEN_min==0.0 && j!=1)
               GREEN_min=K2-X2*(K3-K2)/(X3-X2);
            if(K3>GREEN_max) GREEN_max=K3;
            if(K3<GREEN_min) GREEN_min=K3;
            }

         converge_ck=converge(n,range,X1,Y1);
         if(converge_ck > 0)
            {
            _lineto(X1*max,-Y1*aspect_ratio*max);
            goto NEXT;
            }
         if(fabs(X3)>X_max)
            {
            if(forcolor2==GREEN) infinite_green=1;
            if(forcolor2==RED) infinite_red=1;
            if(j<10)
               {
               midx=X2;
               midy=Y2;
               }

            if((Y3-midy)==0.0 && X3<0.0)
               theta=pi;
            else if((Y3-midy)==0.0)
               theta=0.0;
            else
               theta=atan2((Y3-midy),(X3-midx));
            arrow(X3*max,-Y3*aspect_ratio*max,theta,forcolor2,aspect_ratio);
            goto NEXT;
            }
         if(Y3>=Y_max)
            {
            if(forcolor2==GREEN) infinite_green=1;
            if(forcolor2==RED) infinite_red=1;
            if(j<10)
               {
               midx=X1;
               midy=Y1;
               }
            if((X3-midx)==0.0)
               theta=sign(Y3)*pi/2.0;
            else
               theta=atan2((Y2-midy),(X2-midx));
```

```
                arrow(X3*max,-Y3*aspect_ratio*max,theta,forcolor2,aspect_ratio);
                goto NEXT;
                }
             if(Y3<=Y_min)
                {
                if(forcolor2==GREEN) infinite_green=1;
                if(forcolor2==RED) infinite_red=1;
                if(j<10)
                   {
                   midx=X1;
                   midy=Y1;
                   }

                if((X3-midx)!=0.0)
                   theta=atan2((Y3-midy),(X3-midx));
                else
                   theta=sign(Y3)*pi/2.0;
                arrow(X3*max,-Y3*aspect_ratio*max,theta,forcolor2,aspect_ratio);
                goto NEXT;
                }
             _lineto(X3*max,-Y3*aspect_ratio*max);

             K1=K2;
             K2=K3;
             X1=X2;
             X2=X3;
             Y1=Y2;
             Y2=Y3;
             Yp1=Yp2;
             Yp2=Yp3;
             }
          }
NEXT:
          if(j>800)
             {
             delta_s=delta_s/2;
             goto BEGIN;
             k++;
             }
          if(k>5)
             break;
          }

          if(infinite_red==1 && infinite_green!=1 && RED_min!=0.0 )
             {
             _settextposition(vc.numtextrows-1,2);
             green_length=sprintf(green_string,"Stable Gain Range:   0<K<%-.3g"
                ,RED_min);
             _outtext(green_string);
             _settextposition(vc.numtextrows,2);
             red_length=sprintf(red_string,"Unstable Gain Range: %-.3g<K<\354"
                ,RED_min);
             _outtext(red_string);
             }

          else if(infinite_red==1 && infinite_green!=1 && GREEN_min>0.0)
             {
             green_length=sprintf(green_string,"Stable Gain Range:   %-.3g<K<%-.3g"
                ,GREEN_min,GREEN_max);
             _settextposition(vc.numtextrows-1,2);
             _outtext(green_string);
             red_length=sprintf(red_string,"Unstable Gain Range: 0<K<%-.3g  %-.3g<K<\354"
                ,GREEN_min,GREEN_max);
             _settextposition(vc.numtextrows,2);
             _outtext(red_string);
             }

          else if(infinite_green==1 && infinite_red!=1 && RED_min>0.0)
             {
             green_length=sprintf(green_string,"Stable Gain Range:   0<K<%-.3g
%-.3g<K<\354"
                ,RED_min,RED_max);
```

```
   _settextposition(vc.numtextrows-1,2);
   _outtext(green_string);
   red_length=sprintf(red_string,"Unstable Gain Range: %-.3g<K<%-.3g"
      ,RED_min,RED_max);
   _settextposition(vc.numtextrows,2);
   _outtext(red_string);
   }

else if(infinite_green==1 && infinite_red==1 && RED_min>0.0)
   {
   green_length=sprintf(green_string,"Stable Gain Range:   0<K<%-.3g"
      ,RED_min);
   _settextposition(vc.numtextrows-1,2);
   _outtext(green_string);
   red_length=sprintf(red_string,"Unstable Gain Range: %-.3g<K<\354",
      RED_min);
   _settextposition(vc.numtextrows,2);
   _outtext(red_string);
   }
else if(infinite_green==1 && infinite_red!=1 && RED_min==0.0 && RED_max!=0.0)
   {
   green_length=sprintf(green_string,"Stable Gain Range:   %-.3g<K<\354"
      ,RED_max);
   _settextposition(vc.numtextrows-1,2);
   _outtext(green_string);
   red_length=sprintf(red_string,"Unstable Gain Range: 0<K<%-.3g"
      ,RED_max);
   _settextposition(vc.numtextrows,2);
   _outtext(red_string);
   }

else if(infinite_green==1 && infinite_red!=1 && RED_max!=0.0)
   {
   green_length=sprintf(green_string,"Stable Gain Range:   %-.3g<K<\354"
      ,RED_max);
   _settextposition(vc.numtextrows-1,2);
   _outtext(green_string);
   red_length=sprintf(red_string,"Unstable Gain Range: 0<K<%-.3g",RED_max);
   _settextposition(vc.numtextrows,2);
   _outtext(red_string);
   }

else if(infinite_green!=1 && infinite_red!=1 && RED_min!=0.0)
   {
   green_length=sprintf(green_string,"Stable Gain Range:   0<K<%-.3g"
      ,(GREEN_max+RED_min)/2);
   _settextposition(vc.numtextrows-1,2);
   _outtext(green_string);
   red_length=sprintf(red_string,"Unstable Gain Range: %-.3g<K<\354"
      ,(GREEN_max+RED_min)/2,RED_max);
   _settextposition(vc.numtextrows,2);
   _outtext(red_string);
   }

else if(infinite_green!=1 && infinite_red!=1 && RED_max!=0.0)
   {
   green_length=sprintf(green_string,"Stable Gain Range:   %-.3g<K<%-.3g",
      GREEN_min,(GREEN_max+RED_min)/2);
   _settextposition(vc.numtextrows-1,2);
   _outtext(green_string);
   red_length=sprintf(red_string,"Unstable Gain Range: %-.3g<K<%-.3g"
      ,(GREEN_max+RED_min)/2,RED_max);
   _settextposition(vc.numtextrows,2);
   _outtext(red_string);
   }
else if(RED_max==0.0)
   {
   green_length=sprintf(green_string,"Stable Gain Range: 0<K<\354");
   _settextposition(vc.numtextrows-1,2);
   _outtext(green_string);
   }
```

```
        else
           {
           red_length=sprintf(red_string,"Unstable Gain Range: 0<K<%\354");
           _settextposition(vc.numtextrows-1,2);
           _outtext(red_string);
           }

        _setcolor(forcolor1);
        _setlogorg ( 0, 0 );

        if(green_length>=red_length)
           _rectangle(_GBORDER,0,min,(green_length+2)*8,(vc.numypixels-1));
        else
           _rectangle(_GBORDER,0,min,(red_length+2)*8,(vc.numypixels-1));

}
/******************* START OF MAIN LOCUS PLOT ROUTINE ***********************/

/***************************  EQUATION Y PROGRAM  ***************************/
/*                                                                          */
/*    This routine computes the equation for Y as given in the text,        */
/*    "Modern Control Systems, A Manual of Design Methods", by              */
/*    John A. Borrie, 1986 Prentice-Hall International (UK).                */
/*    The equation Y is given in equations (2.6) to (2.10) pages            */
/*    103 to 105.                                                           */
/*                                                                          */
/*********************  START OF EQUATION FOR Y ROUTINE  ********************/
double  Y(int m, int n, double X, double Y)

{
      double  sigma_diff,
              omega_diff,
              Xp_new;

      int     i,
              k;

      sigma_diff=X-sigmap[1];
      omega_diff=Y-omegap[1];

      Xp=sigma_diff/(sigma_diff*sigma_diff + omega_diff*omega_diff);
      Yp=-omega_diff/(sigma_diff*sigma_diff + omega_diff*omega_diff);

      for(k=2;k<=m;k++)
              {
              sigma_diff=X-sigmap[k];
              omega_diff=Y-omegap[k];
              Xp_new=(sigma_diff*Xp + omega_diff*Yp)/(sigma_diff*sigma_diff
                    + omega_diff*omega_diff);
              Yp=(sigma_diff*Yp - omega_diff*Xp)/(sigma_diff*sigma_diff
                    + omega_diff*omega_diff);
              Xp=Xp_new;
              }

      for(i=1;i<=n;i++)
              {
              sigma_diff=X-sigmaz[i];
              omega_diff=Y-omegaz[i];
              Xp_new=sigma_diff*Xp - omega_diff*Yp;
              Yp=sigma_diff*Yp + omega_diff*Xp;
              Xp=Xp_new;
              }
      return(Yp);
}
/************************  END OF EQUATION Y PROGRAM  ************************/

/*****************************  SLOPE PROGRAM  *****************************/
/*                                                                          */
/*      This routine computes the slope m as give     the text,             */
/*      "Modern Control Systems, A Manual of Desi     thods", by            */
/*      John A. Borrie, 1986 Prentice-Hall Interne    al (UK).              */
/*      The slope equation is given in equation (2.16) on page 108.         */
```

```
/*                                                                      */
/************************  START OF SLOPE ROUTINE  ************************/
double  slope(int zeros, int poles, double sigma, double omega)

{
      double  m,
              rpk=0.0,
              rzi=0.0,
              omega_p=0.0,
              omega_z=0.0,
              sigma_p=0.0,
              sigma_z=0.0;

      int     i,
              k;

      for(k=1;k<=poles;k++)
         {
         rpk=(omega-omegap[k])*(omega-omegap[k])+(sigma-sigmap[k])*(sigma-sigmap[k]);
         omega_p=omega_p+(omega-omegap[k])/rpk;
         sigma_p=sigma_p+(sigma-sigmap[k])/rpk;
         }

      for(i=1;i<=zeros;i++)
         {
         rzi=(omega-omegaz[i])*(omega-omegaz[i])+(sigma-sigmaz[i])*(sigma-sigmaz[i]);
         omega_z=omega_z+(omega-omegaz[i])/rzi;
         sigma_z=sigma_z+(sigma-sigmaz[i])/rzi;
         }

      m=(omega_p-omega_z)/(sigma_p-sigma_z);
      return(m);
}
/*************************  END OF SLOPE PROGRAM  *************************/

/*****************************  CONVERGE PROGRAM  *****************************/
/*                                                                      */
/*      This routine determines if the (X,Y) has converged to any       */
/*      of the n zeros.  It checks for convergence of plus or minus     */
/*      2*range where range is given in the main program.  If it is     */
/*      within the limit the function returns 1 for true, else it       */
/*      returns 0 for faluse.                                           */
/*                                                                      */
/************************  START OF CONVERGE ROUTINE  ************************/
int converge(int n, double range, double X, double Y)
{
              int i;
              double  crange=2*range;

              for(i=1;i<=n;i++)
              {
              if( fabs(X-sigmaz[i])<=crange && fabs(Y-omegaz[i])<=crange)
              return(1);
              }
              return(0);
}
/************************  END OF CONVERGE ROUTINE  ************************/

/*****************************  SORT PROGRAM  *****************************/
/*                                                                      */
/*      This routine sorts the n poles in lowest to highest order.      */
/*                                                                      */
/************************  START OF SORT ROUTINE  ************************/
void    sort(int n)
{
      double  temp[2];

      int     gap,
              i,
              j;

      for(gap=n/2;gap>0;gap/=2)
```

```
                {
                for(i=gap;i<n;i++)
                {
                for(j=i-gap;j>=0&&poles[j][1]>poles[j+gap][1];j-=gap)
                {
                temp[0]=poles[j][1];
                temp[1]=poles[j][2];
                poles[j][1]=poles[j+gap][1];
                poles[j][2]=poles[j+gap][2];
                poles[j+gap][1]=temp[0];
                poles[j+gap][2]=temp[1];
                }
                }
                }
}
/*************************  END OF SORT PROGRAM  *************************/

/***********************  START OF LOCUS OUTPUT SCREEN  ********************/
/*                                                                        */
/*    This is program plots the root locus of a given transfer function.  */
/*                                                                        */
/*************************  START OF LOCUS ROUTINE  ************************/
void output_screen()
{
   int   i,
         j,
         x,
         xlow,
         xhigh,
         xcenter,
         ycenter,
         step,
         textlength,
         position[10][2];

   float index,
         textx,
         texty,
         textmax,
         fabs1,
         fabs2;

   char  indexstring[10];

   if(!set_video_mode())
      {
      printf("\nWarning! This program doesn't support this machine's video card");
      exit(0);
      }

   _getvideoconfig(&vc);
   aspect_ratio = (float) (8.0 * vc.numypixels)/(6.0 * vc.numxpixels);
   xcenter = vc.numxpixels / 2 - 1;
   ycenter = vc.numypixels / 2 - 1;

   _setcolor(GREEN);
   x=vc.numxpixels/40;
   _moveto(1,1);
   _lineto(x,x*aspect_ratio);
   _moveto(1,x*aspect_ratio);
   _lineto(x,1);
   greenpole=(char *)malloc((unsigned int) _imagesize(0,0,x,
   x*aspect_ratio));
   _getimage(0,0,x,x*aspect_ratio, greenpole);

   xlow=50-x/2;
   xhigh=50+x/2;
   _moveto(50,50);
   _ellipse(_GBORDER,xlow,xlow*aspect_ratio,xhigh,xhigh*aspect_ratio);
   greenzero=(char *)malloc((unsigned int) _imagesize(xlow,xlow*
   aspect_ratio-1,xhigh,xhigh*aspect_ratio+1));
   _getimage(xlow,xlow*aspect_ratio-1,xhigh,xhigh*aspect_ratio+1,greenzero);
```

```
_setcolor(RED);
x=vc.numxpixels/40;
_moveto(1,1);
_lineto(x,x*aspect_ratio);
_moveto(1,x*aspect_ratio);
_lineto(x,1);
redpole=(char *)malloc((unsigned int) _imagesize(0,0,x,x*aspect_ratio));
_getimage(0,0,x,x*aspect_ratio, redpole);

xlow=50-x/2;
xhigh=50+x/2;
_moveto(50,50);
_ellipse(_GBORDER,xlow,xlow*aspect_ratio,xhigh,xhigh*aspect_ratio);
redzero=(char *)malloc((unsigned int) _imagesize(xlow,xlow*aspect_ratio-1,
   xhigh,xhigh*aspect_ratio+1));
_getimage(xlow,xlow*aspect_ratio-1,xhigh,xhigh*aspect_ratio+1,redzero);

_clearscreen(_GCLEARSCREEN);
_setcolor(forcolor1);
_settextcolor(forcolor1);

_settextposition((vc.numtextrows-1),78-maxchar+skipn);
_outtext(nstring);
_settextposition((vc.numtextrows),78-maxchar+skipd);
_outtext(dstring);
_moveto(630-maxchar*(vc.numxpixels/vc.numtextcols),460*aspect_ratio);
_lineto(630,460*aspect_ratio);
min=vc.numypixels-43*aspect_ratio;
_rectangle(_GBORDER,0,min,(vc.numxpixels-1),(vc.numypixels-1));
_moveto((vc.numxpixels-2-(maxchar+2)*vc.numxpixels/vc.numtextcols),min);
_lineto((vc.numxpixels-2-(maxchar+2)*vc.numxpixels/vc.numtextcols),
   (vc.numypixels-1));
_settextposition(1,vc.numtextcols/2+2);
_outtext("jw");
_settextposition(vc.numtextrows/2,79);
_outtext("\345");
_rectangle(_GBORDER,0,0,(vc.numxpixels-1),(vc.numypixels-1));
arrow(xcenter,2,1.5708,forcolor1,aspect_ratio);
arrow(vc.numxpixels-2,ycenter,0,forcolor1,aspect_ratio);
_setlinestyle(0x0101);
_moveto(0,ycenter);
_lineto(vc.numxpixels,ycenter);
_moveto(xcenter,0);
_lineto(xcenter,min);
_setlinestyle(0xFFFF);
_moveto(0,min);
_lineto(vc.numxpixels-2,min);
max=0.0;

for(i=0;i<=n-1;i++)
   {
   fabs1=fabs(poles[i][1]);
   fabs2=fabs(poles[i][2]);
   if(fabs1>fabs(max)) max=fabs1;
   if(fabs2>fabs(max)) max=fabs2;
   }

for(i=0;i<=m-1;i++)
   {
   fabs1=fabs(zeros[i][1]);
   fabs2=fabs(zeros[i][2]);
   if(fabs1>fabs(max)) max=fabs1;
   if(fabs2>fabs(max)) max=fabs2;
   }

textmax=2*max;
                  /*  Set up coord. tic marks */
textx=vc.numtextrows/2+2;
for(i=1;i<=9;i++)
   {
   step=64*i;
```

```
        index=(i-5)*textmax/5;
        textlength=sprintf(indexstring,"%-.3g",index);
        texty=i*vc.numtextcols/10-textlength/2+1;
        if(i==5)
         continue;
        else
         {
         _settextposition(textx,texty);
         _outtext(indexstring);
         _moveto(step,ycenter);
         _lineto(step,(ycenter+10*aspect_ratio));
         }
        }
     texty=vc.numtextcols/2+3;
     for(i=1;i<=6;i++)
        {
        step=ycenter+(192-i*64)*aspect_ratio;
        index=(i-3)*textmax/5;
        sprintf(indexstring,"%-.3g",index);
        textx=step*vc.numtextrows/vc.numypixels+1;
        if(i==3)
         continue;
        else
         {
         _settextposition(textx,texty);
         _outtext(indexstring);
         _moveto(xcenter,step);
         _lineto(xcenter+10,step);
         }
        }

     max=vc.numxpixels/4/max;
     x=x/2;
     _setcolor(forcolor2);
     _setlogorg ( xcenter, ycenter );
     for(i=0;i<=n-1;i++)
        {
        position[i][0]=max*poles[i][1]-x;
        position[i][1]=(max*poles[i][2]-x)*aspect_ratio;
        for(j=0;j<=i;j++)
         if(j!=i)
         if(position[i][0] == position[j][0] && position[i][1] == position[j][1])
            position[i][0]=position[i][0]+3;

         if(position[i][0]<=0.0)
            _putimage(position[i][0],position[i][1],greenpole, _GXOR);
         else
            _putimage(position[i][0],position[i][1],redpole, _GXOR);
        }

     for(i=0;i<=m-1;i++)
        {
        position[i][0]=max*zeros[i][1]-x;
        position[i][1]=(max*zeros[i][2]-x)*aspect_ratio;
        for(j=0;j<=i;j++)
         if(j!=i)
            if(position[i][0] == position[j][0] && position[i][1] == position[j][1])
               position[i][0]=position[i][0]+3;

         if(position[i][0]<=0.0)
            _putimage(position[i][0],position[i][1]-1,greenzero, _GXOR);
         else
            _putimage(position[i][0],position[i][1]-1,redzero, _GXOR);
        }
}
/*********************  END OF LOCUS OUTPUT PROGRAM  *********************/

/*****************************  END OF LOCUS FILE  *****************************/
```

APPENDIX D

BODE PLOT SOURCE CODE

```
/************************ START OF BODE ROUTINE ******************************/
/*                                                                           */
/*            This program computes and displays Bode plots.                 */
/*                                                                           */
/************************  START GLOBAL VARIABLES  ***************************/
#include        "variables.h"                    /* Include variable list */
void            bode_output();                      /* Output subroutine */
double          magnitude_max,                  /* Maximum magnitude value */
                magnitude_min,                 /* Minimum magnitude value */
                phase_max,                        /* Maximum phase value */
                phase_min,                        /* Minimum phase value */
                points[640][3],                  /* Plotting points data */
                points2[640][3],                 /* Plotting points data */
                pi=3.14159265,
                  phase_margin,
                  gain_margin,
                gain_cross,
                phase_cross;


float             omega1,                          /* Initial lower frequency */
                omegau,                         /* Initial upper frequency */
                k_initial;                         /* Open-loop gain value */

int               gain_check=0,                   /* Check for gain crossover */
                phase_check=0;                 /* Check for phase crossover */

/************************  END GLOBAL VARIABLES  *****************************/

/********************  START OF MAIN BODE ROUTINE  ***************************/
void bode(int n,int m)
{

   int     l,                                                 /* Loop counter */
           j,                                                /* Loop counter */
           rv,                                               /* Return value */
           itterations,                 /* Number of frequency itterations */
           step,
           count=1;

   float   k_final,                                      /* Final gain value */
           delta_x,                          /* Horizontal screen step size */
           test;

   double  magnitude=0.0,                             /* Magnitude running total */
           phase=0.0,                                  /* Phase running total */
           omega,                                   /* Present frequency value */
           error=0.00001,                                 /* Allowable error */
           start=0.0,                               /* Graph starting point */
           stop=0.0,                                /* Graph stopping point */
           delta_omega,                             /* Frequency step size */
           temp;                                        /* Temporary value */

char    kbuff[10],
      lowerbuff[10],
      upperbuff[10];

      WINDOWPTR wind1,                                 /* input window handle */
      WINDOWPTR wind2;                                 /* input window handle */
      WIFORM frm;                                       /* input form handle */
      int batrib;                                            /* border atrib */
      int watrib;                                            /* window atrib */
      register i;

   /*
    * Set window attributes:
    *
    *      border - blue/white box
    *      window - white background/black letters
    *
   */
```

```
   batrib = v_setatr(BLACK,WHITE,0,0);
   watrib = v_setatr(BLACK,WHITE,0,0);

   /*
    * Open window at 0,0 - 78 cells wide and 23 cells high
   */



   batrib = v_setatr(WHITE,BLACK,0,0);
   watrib = v_setatr(WHITE,BLACK,0,0);

   wind2 = wn_open(0,5,4,70,11,watrib,batrib);
   if(!wind2) exit(1);
   wn_titla(wind2," Bode Input Menu ",batrib);

   wn_printf(wind2,"\n            Input the transfer function information \n\n");
   wn_printf(wind2,"  Example:\n");
   wn_printf(wind2,"  Input the open-loop transfer function gain, K         : 50\n");
   wn_printf(wind2,"  Input the lower value of the frequency plotting range : 0.01\n");
   wn_printf(wind2,"  Input the upper value of the frequency plotting range : 1000\n");

wn_printf(wind2,"————————————————————————————————————————————————————————————\n
");
   wn_printf(wind2,"  Input the open-loop transfer function gain, K         : \n");
   wn_printf(wind2,"  Input the lower value of the frequency plotting range : \n");
   wn_printf(wind2,"  Input the upper value of the frequency plotting range : ");


   frm=wn_frmopn(4);
   if(!frm) exit(0);

   kbuff[0] = 0;
   lowerbuff[0] = 0;
   upperbuff[0] = 0;

   wn_gfloat(SET,frm,0,wind2,8,57,NSTR,watrib,'
',&k_initial,9,2,0.0,100000,kbuff,NSTR,NSTR);
   wn_gfloat(SET,frm,1,wind2,9,57,NSTR,watrib,'
',&omega1,11,6,0.0,100000,lowerbuff,NSTR,NSTR);
   wn_gfloat(SET,frm,2,wind2,10,57,NSTR,watrib,'
',&omegau,11,3,0.0,1000000,upperbuff,NSTR,NSTR);



   rv=wn_frmget(frm);
   wn_frmcls(frm);
   wn_close(wind2);


   if(!set_video_mode())                              /* Set video mode */
      {
      printf("\nWarning! This program doesn't support this machine's video card");
      exit(0);
      }
   _getvideoconfig(&vc);

   _setcolor(YELLOW);
   _settextcolor(YELLOW);
   _rectangle(_GBORDER,0,0,vc.numxpixels-1,vc.numypixels-1);
   _settextposition(12,28);
   sprintf(string,"Working!      Please wait.");
   _outtext(string);
   _settextposition(14,28);
   sprintf(string,"                                ");
   _outtext(string);
   sprintf(string," ");
   _settextposition(14,28);

   delta_x=vc.numxpixels/vc.numtextcols;
   itterations=delta_x*(vc.numtextcols-12);
   step=itterations/25;
```

```
   test=log10(omega1);
   start=floor(test);
   stop=ceil(log10(omegau));
   omega1=pow(10.0,start);              /* Set omega1 next lower power of 10 */
   omegau=pow(10.0,stop);               /* Set omegau next higher power of 10 */

   delta_omega=(stop-start)/itterations;          /* Set frequency step size */

   magnitude_max=0.0;                              /* Initialize max and min */
   magnitude_min=0.0;                              /*  values to 0           */
   phase_max=0.0;                                  /*                        */
   phase_min=0.0;                                  /*                        */

   for(j=0;j<=itterations;j++)              /* Start of frequency range loop */
      {
      k_final=k_initial;
      magnitude=0.0;                        /* Set magnitude and phase to 0 */
      phase=0.0;                            /* for start of each frequency  */
      omega=pow(10.0,start+delta_omega*j);
      if(j>step*count)
       {
       _outtext(string);
       count++;
       }
      for(l=1;l<=m;l++)
       {
       if(fabs(poles[l-1][1])<error && fabs(poles[l-1][2])<error)  /* Pole at the Origin
*/
          {
          magnitude=magnitude-20.0*log10(omega);
          phase=phase-pi/2;
          }

       else if(fabs(poles[l-1][2])<error)              /* Real Axis Pole */
          {
          k_final=k_final/(-poles[l-1][1]);
          temp=1.0+(omega/(-poles[l-1][1]));
          if(temp!=0.0);
             {
             magnitude=magnitude-20.0*log10(fabs(temp));
             phase=phase-atan2(omega,(-poles[l-1][1]));
             }
          }
       else                                              /* Complex Pole */
          {

magnitude=magnitude-20.0*log10(sqrt(poles[l-1][1]*poles[l-1][1]+(omega-poles[l-1][2])*(om
ega-poles[l-1][2])));
          phase=phase+atan2((omega-poles[l-1][2]),poles[l-1][1]);
          }
       }

      for(l=1;l<=n;l++)
       {
       if(fabs(zeros[l-1][1])<error && fabs(zeros[l-1][2])<error)  /* Zero at the Origin
*/
          {
          magnitude=magnitude+20.0*log10(omega);
          phase=phase+pi/2;
          }
       else if(fabs(zeros[l-1][2])<error)              /* Real Axis Zero */
          {
          k_final=k_final*(-zeros[l-1][1]);
          temp=1.0+(omega/(-zeros[l-1][1]));
          if(temp!=0.0);
             {
             magnitude=magnitude+20.0*log10(fabs(temp));
             phase=phase+atan2(omega,(-zeros[l-1][1]));
             }
          }
       else                                              /* Complex Zero */
```

```
         {

magnitude=magnitude+20.0*log10(sqrt(zeros[1-1][1]*zeros[1-1][1]+(omega-zeros[1-1][2])*(om
ega-zeros[1-1][2])));
         phase=phase-atan2((omega-zeros[1-1][2]),zeros[1-1][1]);
         }
       }
      if(k_final != 1.0 && k_final > 0.0)
       magnitude=magnitude+20.0*log10(k_final);
      phase=phase*180.0/pi;

      points2[j][1]=omega;                          /* Store plotting data */
      points2[j][2]=magnitude;                      /*                     */
      points2[j][3]=phase;                          /*                     */

      while(fabs(phase)>180.0) phase=phase-sign(phase)*360;
      if(magnitude>magnitude_max) magnitude_max=magnitude;
      if(magnitude<magnitude_min) magnitude_min=magnitude;
      if(phase>phase_max) phase_max=phase;
      if(phase<phase_min) phase_min=phase;

      points[j][1]=omega;                           /* Store plotting data */
      points[j][2]=magnitude;                       /*                     */
      points[j][3]=phase;                           /*                     */
   }



   for(j=0;j<=itterations;j++)            /* Start of frequency range loop */
      {
      if(j!=0)
      {
      if(sign(points2[j][2])!=sign(points2[j-1][2]))  /* Check for phase margin */
       {
       phase_margin=180+points2[j-1][3]-points2[j-1][2]*
            (points2[j][3]-points2[j-1][3])/(points2[j][2]-points2[j-1][2]);

       phase_cross=points2[j-1][1]-points2[j-1][2]*
            (points2[j][1]-points2[j-1][1])/(points2[j][2]-points2[j-1][2]);
       phase_check=1;
       }

      if(sign(points2[j-1][3]+180) != sign(points2[j][3]+180)) /* Check for gain margin
*/
       {
       gain_margin=-points2[j-1][2]-(points2[j][2]-points2[j-1][2])*
                    (-180-points2[j-1][3])/(points2[j][3]-points2[j-1][3]);

       gain_cross=(points2[j-1][1]+points2[j][1])/2;
       gain_check=1;
       }
      }
   }

       _outtext(string);


   bode_output();                                 /* Call output routine */
}
/*********************  END OF MAIN BODE ROUTINE  ***************************/

/*********************** START OF OUTPUT SCREEN ******************************/
void bode_output()
{

   int     l,
         j,
         itterations;

   float   divisions,
         tic_start,
         tic_stop,
```

```
      rt_upper=0,
      rt_lower=0,
      delta_rt=0,
      angle_upper=0,
      angle_lower=0,
      delta_angle=0,
      x,
      y,
      y1,
      y2,
      tic_y,
      tic_x,
      step_size,
      text_step_size,
      delta_y,
      delta_x,
      length;

double  slope,
      x_slope,
      y_slope,
      x_high,
      x_low,
      y_high,
      y_low;

char    string[20];

aspect_ratio = (float) (8.0 * vc.numypixels)/(6.0 * vc.numxpixels);
delta_x=vc.numxpixels/vc.numtextcols;
delta_y=vc.numypixels/vc.numtextrows;
_clearscreen(_GCLEARSCREEN);
_setcolor(forcolor1);
x_low=6*delta_x;
x_high=delta_x*(vc.numtextcols-6);
y_high=delta_y;
y_low=delta_y*(vc.numtextrows-4)+delta_y/2+3*aspect_ratio;
itterations=delta_x*(vc.numtextcols-12);

_rectangle(_GBORDER,x_low,y_high,x_high,y_low);
y_high=delta_y*1.5;

/* Gain Heading */
_settextcolor(MAGENTA);
_setcolor(MAGENTA);
_settextposition(0,0);
sprintf(string,"    dB Magnitude");
_outtext(string);
_moveto(delta_x+1,delta_y/2-1);
_lineto(3*delta_x+1,delta_y/2-1);
rt_upper=ceil(magnitude_max);
rt_lower=floor(magnitude_min);
for(;;)
   {
   if(fmod(rt_upper,10.0)!=0.0) rt_upper=rt_upper+1.0;
   else break;
   }
for(;;)
   {
   if(fmod(rt_lower,10.0)!=0.0) rt_lower=rt_lower-1.0;
   else break;
   }
for(;;)
   {
   if(fmod(rt_upper-rt_lower,(vc.numtextrows-5))!=0.0)
    {
    rt_upper=rt_upper+5.0;
    rt_lower=rt_lower-5.0;
    }
   else break;
   }
delta_rt=(rt_upper-rt_lower)/(vc.numtextrows-5);
```

```
tic_stop=6*delta_x;
tic_start=tic_stop-delta_x/2;
for(l=0;l<=(vc.numtextrows-5);l++)
   {
   tic_y=delta_y*(1.5+l)-1;
   _moveto(tic_start,tic_y);
   _lineto(tic_stop,tic_y);
   }
for(l=0;l<=(vc.numtextrows-5);l++)
   {
   _settextposition(l+2,1);
   sprintf(string,"%5g",(rt_upper-delta_rt*l));
   _outtext(string);
   }

x_slope=(x_low-x_high)/(log10(omegal)-log10(omegau));
y_slope=(-delta_y*(vc.numtextrows-5))/(rt_upper-rt_lower);

_setlinestyle(0x0101);
x=x_low;
y=y_high-rt_upper*y_slope;
_moveto(x,y);
x=x_high;
_lineto(x,y);
_setlinestyle(0xFFFF);

x=x_low-(log10(omegal)-log10(points[0][1]))*x_slope;
y=y_high-(rt_upper-points[0][2])*y_slope;
_moveto(x,y);

for(l=1;l<=itterations;l++)
   {
   x=x_low-(log10(omegal)-log10(points[l][1]))*x_slope;
   y=y_high-(rt_upper-points[l][2])*y_slope;
   _lineto(x,y);
   }

/* Phase Heading */
_settextcolor(GREEN);
_setcolor(GREEN);
length=sprintf(string,"Phase    ");
_settextposition(0,vc.numtextcols-length);
_outtext(string);
y=delta_y/2-1;
for(j=(vc.numtextcols-4)*delta_x;j<=(vc.numtextcols-2)*delta_x;j+=+2)
   _setpixel(j,y);
tic_start=delta_x*(vc.numtextcols-6);
tic_stop=tic_start+delta_x/2;
for(l=0;l<=(vc.numtextrows-5);l++)
   {
   tic_y=delta_y*(1.5+l)-1;
   _moveto(tic_start,tic_y);
   _lineto(tic_stop,tic_y);
   }
angle_upper=ceil(phase_max);
angle_lower=floor(phase_min);
for(;;)
   {
   if(fmod(angle_upper,10.0)!=0.0) angle_upper=angle_upper+1.0;
   else break;
   }
for(;;)
   {
   if(fmod(angle_lower,10.0)!=0.0) angle_lower=angle_lower-1.0;
   else break;
   }
for(;;)
   {
   if(fmod((angle_upper-angle_lower),(vc.numtextrows-5))!=0.0)
    {
    angle_upper=angle_upper+5.0;
    angle_lower=angle_lower-5.0;
```

```
    }
   else break;
   }
delta_angle=(angle_upper-angle_lower)/(vc.numtextrows-5);

for(l=0;l<=(vc.numtextrows-5);l++)
   {
   _settextposition(l+2,vc.numtextcols-4);
   sprintf(string,"%-g\370",(angle_upper-delta_angle*l));
   _outtext(string);
   }
x_slope=(x_low-x_high)/(log10(omegal)-log10(omegau));
y_slope=(-delta_y*(vc.numtextrows-5))/(angle_upper-angle_lower);

x=x_low-(log10(omegal)-log10(points[0][1]))*x_slope;
y=y_high-(angle_upper-points[0][3])*y_slope;
_moveto(x,y);
for(l=2;l<=itterations;l+=+2)
   {
   x=x_low-(log10(omegal)-log10(points[l][1]))*x_slope;
   y2=y_high-(angle_upper-points[l][3])*y_slope;
   _setpixel(x,y2);
   if(abs(y2-y)>2 && sign(slope)==sign(y2-y))
    {
    y1=y;
    for(j=1;j<=abs(y2-y)/2;j+=+2)
       {
       y1=y1+sign(y2-y)*2;
       _setpixel(x-1,y1);
       }
    for(j=1;j<=abs(y2-y)/2;j+=+2)
       {
       y1=y1+sign(y2-y)*2;
       _setpixel(x,y1);
       }
    }
   slope=y2-y;
   y=y2;
   }
   _settextcolor(forcolor1);
   _setcolor(forcolor1);
   divisions=log10(omegau)-log10(omegal);
   step_size=(vc.numxpixels-12*delta_x)/divisions;
   text_step_size=(vc.numtextcols-12)/divisions;
   tic_start=delta_y*(vc.numtextrows-4)+delta_y/2+3*aspect_ratio;
   tic_stop=tic_start+delta_x/2;
   for(l=0;l<=divisions;l++)
    {
    tic_x=6*delta_x+step_size*l;
    _moveto(tic_x,tic_start);
    _lineto(tic_x,tic_stop);
    if(l!=divisions)
       {
       for(j=1;j<=8;j++)
          {
          x=x_low-(log10(omegal)-log10(omegal*pow(10,1)*(j+1)))*x_slope;
          _moveto(x,tic_start);
          _lineto(x,tic_stop-delta_x/4);
          }
       }
    length=sprintf(string,"%g",omegal*pow(10,l));
    _settextposition(23,7+text_step_size*l-length/2);
    _outtext(string);
    }

string[0]='\0';
strncpy(string,nstring,25);
length=strlen(string);
for(l=1;l<=length+1;l++)
   nstring[l-1]=string[l];
string[0]='\0';
if(k_initial!=1)
```

```
   {
   sprintf(string,"%g",k_initial);
   strncat(string,nstring,25);
   strncpy(nstring,string,25);
   }
if(k_initial==1 && length<2)
   {
   sprintf(string,"%g",k_initial);
   strncat(string,nstring,25);
   strncpy(nstring,string,25);
   }
length=strlen(nstring);
if(length>maxchar)
   {
   skipd=skipd+(length-maxchar)/2;
   maxchar=length;
   }
else
   skipn=(maxchar-length)/2+2;
_settextposition((vc.numtextrows-1),78-maxchar+skipn);
_outtext(nstring);
_settextposition((vc.numtextrows),78-maxchar+skipd);
_outtext(dstring);
y=460*aspect_ratio;
_moveto(630-maxchar*(vc.numxpixels/vc.numtextcols),y);
_lineto(630,y);
min=vc.numypixels-43*aspect_ratio;
_rectangle(_GBORDER,0,min+2,(vc.numxpixels-1),(vc.numypixels-1));
_moveto((vc.numxpixels-2-(maxchar+2)*vc.numxpixels/vc.numtextcols),min+2);
_lineto((vc.numxpixels-2-(maxchar+2)*vc.numxpixels/vc.numtextcols),(vc.numypixels-1));

length=sprintf(string,"  Bode Plot");
_settextposition(0,(vc.numtextcols-length)/2);
_outtext(string);
x=(vc.numxpixels-(length+1)*delta_x)/2;
_moveto(x,0);
_lineto(x,delta_y);
x=(vc.numxpixels+(length+1)*delta_x)/2-delta_x-2;
_moveto(x,0);
_lineto(x,delta_y);

if(phase_check==1)
   {
   length=sprintf(string,"Phase Margin = %.1g\370  ", phase_margin);
   _settextposition(24,2);
   _outtext(string);

   length=sprintf(string,"@ Freq. = %.1g", phase_cross);
   _settextposition(24,24);
   _outtext(string);
   }
else
   {
   length=sprintf(string,"Phase margin is not defined.");
   _settextposition(24,2);
   _outtext(string);
   }

if(gain_check==1)
   {
   length=sprintf(string,"Gain  Margin = %.1g dB", gain_margin);
   _settextposition(25,2);
   _outtext(string);

   length=sprintf(string,"@ Freq. = %.1g", gain_cross);
   _settextposition(25,24);
   _outtext(string);
   }
else
   {
   length=sprintf(string,"Gain margin is not defined.");
   _settextposition(25,2);
```

```
      _outtext(string);
      }
   _rectangle(_GBORDER,0,0,vc.numxpixels-1,vc.numypixels-1);

}
/************************  END OF OUTPUT SCREEN  ****************************/

/************************  END OF BODE ROUTINE  *****************************/
```

APPENDIX E

MODIFIED WINDOWS SOURCE CODE

```
/****************** START OF MODIFIED WINDOW ROUTINES ********************/
/*                                                                       */
/*    These two routines are modified versions of the popup and formget  */
/*    routines supplied in the windowing package                         */
/*                                                                       */
/*                              The Window BOSS                          */
/*                            by Phil Mongelluzzo                        */
/*                                                                       */
/*                            Revision: 08.15.88                         */
/*                                                                       */
/*                      Star Guidance Consulting, Inc.                   */
/*                              273 Windy Drive                          */
/*                       Waterbury, Connecticut 06705                    */
/*                                                                       */
/*                              (203) 574-2449                           */
/*                                                                       */
/*************************************************************************/
/*
** POPUP - Popup Menu/Window Driver
**
** Copyright (c) 1985-1988 Philip A. Mongelluzzo
** All rights reserved.
*/

#include "windows.h"                    /* window header */

#undef UARROW                           /* dont argue with windows.h */
#undef DARROW
#undef LARROW
#undef RARROW
#undef BS
#undef DEL
#undef RET
#undef ESC


#define UARROW  0x48                    /* define key codes */
#define DARROW  0x50
#define LARROW  0x4b
#define RARROW  0x4d
#define BS      0x0e
#define DEL     0x53
#define RET     0x1c
#define ESC     0x01
#define SPACE   0x39

#define ZERO    0x00
#define ONE     0x02
#define TWO     0x03
#define THREE   0x04
#define FOUR    0x05
#define FIVE    0x06
#define SIX     0x07
#define SEVEN   0x08
#define EIGHT   0x09


#define v_setrev(atrib) ((atrib&0x88) ¦ ((atrib>>4)&0x07) ¦ ((atrib<<4)&0x70) )

/*
** Popup structure templates
*/

  struct m     {                      /* menu item template */
    int r;                            /* row */
    int c;                            /* col */
    char *t;                          /* text */
    int rv;                           /* return value */
  };
```

```
  struct pmenu {                          /* popup menu structure */
    WINDOWPTR wpsave;                     /* place to hold window id */
    int winopn;                           /* leave window open flag */
    int lndx;                             /* last index */
    int fm;                               /* first menu item index */
    int lm;                               /* last menu item index */
    struct mitem scrn[25];                /* a bunch of menu items */
  };

/*
*********
* popup *
*********
*/

popup(page,row,col,width,height,atrib,batrib,mx,cflag)
int page;                                 /* video page */
int row, col;                             /* window - upper row/col */
int width, height;                        /* window - height & width */
struct pmenu *mx;                         /* pointer to popup menu struct */
int cflag;                                /* close on return strike flag */
int atrib, batrib;                        /* attributes - window & border */
{


int i;                                    /* scratch integer */
WINDOWPTR w;                              /* window pointer */
int c;                                    /* key scan code,,char */


  if(mx->winopn) {                        /* window is still open */
    goto d0;                              /* enter processing loop */
  }
  mx->lndx = -1;                          /* set index out of range */
  w = wn_open(page, row, col, width, height, atrib, batrib);
  wn_sync(w,FALSE);                       /* sync cursor */
  mx->wpsave = w;                         /* save pointer */
  if (w == NULL) return(99);              /* out of mem or just fubar */
  mx->winopn = TRUE;                      /* say window is open */

  i = 0;                                  /* init index */

  while(mx->scrn[i].r != 99) {            /* for as long as we have data */
    wn_putsa(w, mx->scrn[i].r, mx->scrn[i].c, mx->scrn[i].t, atrib);
    i++;
  }

d0:
  w = mx->wpsave;                         /* restore pointer */
  i = mx->lndx;                           /* BLINDLY assume that we're back */
  if(i < mx->fm) i = mx->fm;              /* and reset if "i" is now out of */
  if(i > mx->lm) i = mx->fm;              /* range - (dumb, but it works) */
  while(TRUE) {                           /* till we exit */
    wn_putsa(w, mx->scrn[i].r, mx->scrn[i].c, mx->scrn[i].t, v_setrev(atrib));
    c = v_getch() >>8;

    if(c == ESC) {                        /* ESC (user wants out) - swk */
      if(cflag) {                         /* close window on return strike ? */
      wn_close(w);                       /* close the window */
      mx->winopn = FALSE;                /* say window is closed */
      }
      mx->lndx = i;                       /* remember last indx */
      return(4);                          /* return with rv */
    }
    if(c == RET) {                        /* swk RETURN */
      if(cflag) {                         /* close window on return strike ? */
        wn_close(w);                      /* close the window */
        mx->winopn = FALSE;               /* say window is closed */
      }
      mx->lndx = i;                       /* remember last indx */
      return(mx->scrn[i].rv);             /* return with rv */
    }
```

```
    if(c == ONE) {                      /* swk RETURN */
      if(cflag) {                       /* close window on return strike ? */
      wn_close(w);                     /* close the window */
      mx->winopn = FALSE;              /* say window is closed */
      }
      mx->lndx = i;                     /* remember last indx */
      return(1);                        /* return with rv */
    }
    if(c == TWO) {                      /* swk RETURN */
      if(cflag) {                       /* close window on return strike ? */
      wn_close(w);                     /* close the window */
      mx->winopn = FALSE;              /* say window is closed */
      }
      mx->lndx = i;                     /* remember last indx */
      return(2);                        /* return with rv */
    }
    if(c == THREE) {                     /* swk RETURN */
      if(cflag) {                       /* close window on return strike ? */
      wn_close(w);                     /* close the window */
      mx->winopn = FALSE;              /* say window is closed */
      }
      mx->lndx = i;                     /* remember last indx */
      return(3);                        /* return with rv */
    }
    if(c == FOUR) {                      /* swk RETURN */
      if(cflag) {                       /* close window on return strike ? */
      wn_close(w);                     /* close the window */
      mx->winopn = FALSE;              /* say window is closed */
      }
      mx->lndx = i;                     /* remember last indx */
      return(4);                        /* return with rv */
    }
    if(c == DARROW) c = SPACE;          /* swk conversion */
    if(c == RARROW) c = SPACE;          /* swk conversion */
    if(c == LARROW) c = BS;             /* swk conversion */
    if(c == UARROW) c = BS;             /* swk conversion */
    if(c == SPACE || c == DEL || c == BS) {
      wn_putsa(w, mx->scrn[i].r, mx->scrn[i].c, mx->scrn[i].t, atrib);
      if(c == SPACE)                    /* foreward ?? */
        i++;                            /* bump index */
      else
        i--;                            /* decrement */
      if(i < mx->fm) i = mx->lm;        /* wrap on either */
      if(i > mx->lm) i = mx->fm;        /* end */
    }                                   /* endif */
  }                                     /* end while */
  wn_close(w);                          /* bye bye */
  mx->winopn = FALSE;                   /* say window is closed */
  return(99);                           /* nothing selected */
}

#ifdef COMMENTS
/*
**********                              /* quick popup... */
* qpopup *                             /* this is a hack of popup */
**********                              /* but the code fragment */
*/                                      /* could be of value */
#endif

WINDOWPTR qpopup(page,row,col,width,height,atrib,batrib,mx)
int page;                               /* video page */
int row, col;                           /* window - upper row/col */
int width, height;                      /* window - height & width */
int atrib, batrib;                      /* atributes - window & border */
struct pmenu *mx;                       /* pointer to text struct */
{
int i;                                  /* scratch integer */
WINDOWPTR w;                            /* window pointer */

  w = wn_open(page, row, col, width, height, atrib, batrib);

  i = 0;                                /* init index */
```

```
  while(mx->scrn[i].r != 99) {          /* for as long as we have data */
    wn_puts(w, mx->scrn[i].r, mx->scrn[i].c, mx->scrn[i].t);
    i++;
  }
  return(w);
}

/* End */


/*
** The Window BOSS's Data Clerk
** Copyright (c) 1988 - Philip A. Mongelluzzo
** All rights reserved.
**
** wn_frmget - Get (read) data entry form
**
** Copyright (c) 1988 - Philip A. Mongelluzzo
** All rights reserved.
**
*/

                /* standard stuff */

/*
*************
* wn_frmget *
*************
*/

/*
** wn_frmget(frm)
**
**    int     nfields - number of fields in form.
**
** RETURNS:
**
**    TRUE  - indicating all fields of the form in question have been
**            fetched and verified (where required).
**
**      or
**
**    Never Returns!!
**
** NOTES:
**
**    This code can be used as is or can be customized to suite each
**    applications need.  The section to customize is at the tail end
**    of this file.  As distributed, logic diplays all prompts and
**    display fields, positions to the first field, performs data
**    entry on a field by field basis from the first to the last
**    (allowing editing along the way), asks for a confirmation to
**    accept the fields on the form after the last field is entered,
**    either accepts the form or drops into edit mode for all the
**    fields on the form starting at the first field.
**
**    wn_frmget will not return until all data has been entered and
**    verified (where required).
**
**    This routine must be called after wn_frmopn, and before wn_frmcls.
**
*/

/*
*************
* wn_frmget *
*************
*/

#define FLD frm[indx]                   /* some shorthand */

wn_frmget2(frm)                          /* input form processor */
```

```
WIFORM frm;                             /* array of field pointers */
{
int indx;                               /* input field index */
int rv;                                 /* wn_g???'s return value */
int c;                                  /* scratch */
WINDOWPTR wn;                           /* my OWN window */

static char *msg = "[Press ENTER to Accept, any other key to Edit]";

  indx = 0;                             /* set index */
  while(TRUE) {                         /* display output fields */
    if(FLD->pself != (char *)FLD)       /* corrupted memory check */
      wns_ierr("wm_frmget@wn_puts");    /* die if memory is mangled */
    if(!FLD->fcode) break;              /* all done with output fields */
    wn_puts(FLD->wn, FLD->row, FLD->col, FLD->prmpt);
    indx++;                             /* bump index */
  }

begin:
  indx = 0;                             /* reset index */
  while(TRUE) {                         /* fetch input fields */
    if(!FLD->fcode) break;              /* all done */
    switch (FLD->fcode) {               /* based on function code */
      case GDATE:
      if(FLD->pself != (char *)FLD)   /* corrupted memory check */
        wns_ierr("wm_frmget@GDATE");
      rv = wn_gdate(XEQ,NFRM,NFLD,FLD->wn, FLD->row,
                    FLD->col+strlen(FLD->prmpt), NSTR,
                    FLD->atrib, FLD->fill, FLD->v1.vip, FLD->v2.vip,
                    FLD->v3.vip, FLD->v4.vcp,
                    FLD->v5.vcp,FLD->v6.vcp);
      break;
      case GTIME:
      if(FLD->pself != (char *)FLD)   /* corrupted memory check */
        wns_ierr("wm_frmget@GTIME");
      rv = wn_gtime(XEQ,NFRM,NFLD,FLD->wn, FLD->row,
                    FLD->col+strlen(FLD->prmpt), NSTR,
                    FLD->atrib, FLD->fill, FLD->v1.vip, FLD->v2.vip,
                    FLD->v3.vip, FLD->v4.vcp,
                    FLD->v5.vcp,FLD->v6.vcp);
      break;
      case GINT:
      if(FLD->pself != (char *)FLD)   /* corrupted memory check */
        wns_ierr("wm_frmget@GINT");
      rv = wn_gint(XEQ,NFRM,NFLD,FLD->wn, FLD->row,
                   FLD->col+strlen(FLD->prmpt), NSTR,
                   FLD->atrib, FLD->fill, FLD->v1.vip, FLD->v2.vi,
                   FLD->v3.vi, FLD->v4.vi, FLD->v5.vcp,
                   FLD->v6.vcp,FLD->v7.vcp);
      break;
      case GUINT:
      if(FLD->pself != (char *)FLD)   /* corrupted memory check */
        wns_ierr("wm_frmget@GUINT");
      rv = wn_guint(XEQ,NFRM,NFLD,FLD->wn, FLD->row,
                    FLD->col+strlen(FLD->prmpt), NSTR,
                    FLD->atrib, FLD->fill, FLD->v1.vuip, FLD->v2.vi,
                    FLD->v3.vui, FLD->v4.vui, FLD->v5.vcp,
                    FLD->v6.vcp,FLD->v7.vcp);
      break;
      case GLONG:
      if(FLD->pself != (char *)FLD)   /* corrupted memory check */
        wns_ierr("wm_frmget@GLONG");
      rv = wn_glong(XEQ,NFRM,NFLD,FLD->wn, FLD->row,
                    FLD->col+strlen(FLD->prmpt), NSTR,
                    FLD->atrib, FLD->fill, FLD->v1.vlp, FLD->v2.vi,
                    FLD->v3.vl, FLD->v4.vl, FLD->v5.vcp,
                    FLD->v6.vcp,FLD->v7.vcp);
      break;
      case GFLOAT:
      if(FLD->pself != (char *)FLD)   /* corrupted memory check */
        wns_ierr("wm_frmget@GFLOAT");
      rv = wn_gfloat(XEQ,NFRM,NFLD,FLD->wn, FLD->row,
```

```
                    FLD->col+strlen(FLD->prmpt), NSTR,
                    FLD->atrib, FLD->fill, FLD->v1.vfp, FLD->v2.vi,
                    FLD->v3.vi, FLD->v4.vf, FLD->v5.vf, FLD->v6.vcp,
                    FLD->v7.vcp,FLD->v8.vcp);
      break;
      case GPHONE:
      if(FLD->pself != (char *)FLD)   /* corrupted memory check */
        wns_ierr("wm_frmget@GPHONE");
      rv = wn_gphone(XEQ,NFRM,NFLD,FLD->wn, FLD->row,
                    FLD->col+strlen(FLD->prmpt), NSTR,
                    FLD->atrib, FLD->fill, FLD->v1.vip, FLD->v2.vip,
                    FLD->v3.vip, FLD->v4.vcp,
                    FLD->v5.vcp,FLD->v6.vcp);
      break;
      case GTEXT:
      if(FLD->pself != (char *)FLD)   /* corrupted memory check */
        wns_ierr("wm_frmget@GTEXT");
      rv = wn_gtext(XEQ,NFRM,NFLD,FLD->wn, FLD->row,
                    FLD->col+strlen(FLD->prmpt), NSTR,
                    FLD->atrib, FLD->fill, FLD->v1.vi, FLD->v2.vcp,
                    FLD->v3.vcp,FLD->v4.vcp);
      break;
      case GBOOL:
      if(FLD->pself != (char *)FLD)   /* corrupted memory check */
        wns_ierr("wm_frmget@GBOOL");
      rv = wn_gbool(XEQ,NFRM,NFLD,FLD->wn, FLD->row,
                    FLD->col+strlen(FLD->prmpt), NSTR,
                    FLD->atrib, FLD->fill, FLD->v1.vip, FLD->v2.vcp,
                    FLD->v3.vcp,FLD->v4.vcp);
      break;
      case DTEXT:                       /* display text */
      if(FLD->pself != (char *)FLD)   /* corrupted memory check */
        wns_ierr("wm_frmget@DTEXT");
      rv = wn_dtext(XEQ,NFRM,NFLD,FLD->wn, FLD->row, FLD->col, FLD->prmpt);
      break;
    }                                   /* end switch(frm.code) */
    if(rv != BKTAB && rv != UARROW)     /* previous field ?? */
      indx++;                           /* no, just bump index */
    else if(rv == ESC)
      return(0);
    else {                              /* previous field request */
      indx--;                           /* decrement index */
      if(indx <= 0) indx = 0;           /* but dont be stupid! */
    }
  }

/*****************************************************************************
*                                                                            *
*                  CUSTOMIZE THE SECTION BELOW ONLY                          *
*                                                                            *
*****************************************************************************/

}

/* End */
endif;


/*
** The Window BOSS's Data Clerk
** Copyright (c) 1988 - Philip A. Mongelluzzo
** All rights reserved.
**
** wn_frmget - Get (read) data entry form
**
** Copyright (c) 1988 - Philip A. Mongelluzzo
** All rights reserved.
**
*/


/*
```

```
*************
* wn_frmget *
*************
*/

/*
** wn_frmget(frm)
**
**    int     nfields - number of fields in form.
**
** RETURNS:
**
**    TRUE  - indicating all fields of the form in question have been
**            fetched and verified (where required).
**
**      or
**
**    Never Returns!!
**
** NOTES:
**
**    This code can be used as is or can be customized to suite each
**    applications need.  The section to customize is at the tail end
**    of this file.  As distributed, logic diplays all prompts and
**    display fields, positions to the first field, performs data
**    entry on a field by field basis from the first to the last
**    (allowing editing along the way), asks for a confirmation to
**    accept the fields on the form after the last field is entered,
**    either accepts the form or drops into edit mode for all the
**    fields on the form starting at the first field.
**
**    wn_frmget will not return until all data has been entered and
**    verified (where required).
**
**    This routine must be called after wn_frmopn, and before wn_frmcls.
**
*/

/*
*************
* wn_frmget *
*************
*/

#define FLD frm[indx]                   /* some shorthand */

wn_frmget(frm)                          /* input form processor */
WIFORM frm;                             /* array of field pointers */
{
int indx;                               /* input field index */
int rv;                                 /* wn_g???'s return value */
int c;                                  /* scratch */
WINDOWPTR wn;                           /* my OWN window */

static char *msg = "[Press ENTER to Accept, any other key to Edit]";

  indx = 0;                             /* set index */
  while(TRUE) {                         /* display output fields */
    if(FLD->pself != (char *)FLD)       /* corrupted memory check */
      wns_ierr("wn_frmget@wn_puts");    /* die if memory is mangled */
    if(!FLD->fcode) break;              /* all done with output fields */
    wn_puts(FLD->wn, FLD->row, FLD->col, FLD->prmpt);
    indx++;                             /* bump index */
  }

begin:
  indx = 0;                             /* reset index */
  while(TRUE) {                         /* fetch input fields */
    if(!FLD->fcode) break;              /* all done */
    switch (FLD->fcode) {               /* based on function code */
      case GDATE:
        if(FLD->pself != (char *)FLD)   /* corrupted memory check */
```

```
    wns_ierr("wm_frmget@GDATE");
  rv = wn_gdate(XEQ,NFRM,NFLD,FLD->wn, FLD->row,
                FLD->col+strlen(FLD->prmpt), NSTR,
                FLD->atrib, FLD->fill, FLD->v1.vip, FLD->v2.vip,
                FLD->v3.vip, FLD->v4.vcp,
                FLD->v5.vcp,FLD->v6.vcp);
  break;
case GTIME:
  if(FLD->pself != (char *)FLD)   /* corrupted memory check */
    wns_ierr("wm_frmget@GTIME");
  rv = wn_gtime(XEQ,NFRM,NFLD,FLD->wn, FLD->row,
                FLD->col+strlen(FLD->prmpt), NSTR,
                FLD->atrib, FLD->fill, FLD->v1.vip, FLD->v2.vip,
                FLD->v3.vip, FLD->v4.vcp,
                FLD->v5.vcp,FLD->v6.vcp);
  break;
case GINT:
  if(FLD->pself != (char *)FLD)   /* corrupted memory check */
    wns_ierr("wm_frmget@GINT");
  rv = wn_gint(XEQ,NFRM,NFLD,FLD->wn, FLD->row,
               FLD->col+strlen(FLD->prmpt), NSTR,
               FLD->atrib, FLD->fill, FLD->v1.vip, FLD->v2.vi,
               FLD->v3.vi, FLD->v4.vi, FLD->v5.vcp,
               FLD->v6.vcp,FLD->v7.vcp);
  break;
case GUINT:
  if(FLD->pself != (char *)FLD)   /* corrupted memory check */
    wns_ierr("wm_frmget@GUINT");
  rv = wn_guint(XEQ,NFRM,NFLD,FLD->wn, FLD->row,
                FLD->col+strlen(FLD->prmpt), NSTR,
                FLD->atrib, FLD->fill, FLD->v1.vuip, FLD->v2.vi,
                FLD->v3.vui, FLD->v4.vui, FLD->v5.vcp,
                FLD->v6.vcp,FLD->v7.vcp);
  break;
case GLONG:
  if(FLD->pself != (char *)FLD)   /* corrupted memory check */
    wns_ierr("wm_frmget@GLONG");
  rv = wn_glong(XEQ,NFRM,NFLD,FLD->wn, FLD->row,
                FLD->col+strlen(FLD->prmpt), NSTR,
                FLD->atrib, FLD->fill, FLD->v1.vlp, FLD->v2.vi,
                FLD->v3.vl, FLD->v4.vl, FLD->v5.vcp,
                FLD->v6.vcp,FLD->v7.vcp);
  break;
case GFLOAT:
  if(FLD->pself != (char *)FLD)   /* corrupted memory check */
    wns_ierr("wm_frmget@GFLOAT");
  rv = wn_gfloat(XEQ,NFRM,NFLD,FLD->wn, FLD->row,
                 FLD->col+strlen(FLD->prmpt), NSTR,
                 FLD->atrib, FLD->fill, FLD->v1.vfp, FLD->v2.vi,
                 FLD->v3.vi, FLD->v4.vf, FLD->v5.vf, FLD->v6.vcp,
                 FLD->v7.vcp,FLD->v8.vcp);
  break;
case GPHONE:
  if(FLD->pself != (char *)FLD)   /* corrupted memory check */
    wns_ierr("wm_frmget@GPHONE");
   r = wn_gphone(XEQ,NFRM,NFLD,FLD->wn, FLD->row,
                 FLD->col+strlen(FLD->prmpt), NSTR,
                 FLD->atrib, FLD->fill, FLD->v1.vip, FLD->v2.vip,
                 FLD->v3.vip, FLD->v4.vcp,
                 FLD->v5.vcp,FLD->v6.vcp);
  break;
case GTEXT:
  if(FLD->pself != (char *)FLD)   /* corrupted memory check */
    wns_ierr("wm_frmget@GTEXT");
  rv = wn_gtext(XEQ,NFRM,NFLD,FLD->wn, FLD->row,
                FLD->col+strlen(FLD->prmpt), NSTR,
                FLD->atrib, FLD->fill, FLD->v1.vi, FLD->v2.vcp,
                FLD->v3.vcp,FLD->v4.vcp);
  break;
case GBOOL:
  if(FLD->pself != (char *)FLD)   /* corrupted memory check */
    wns_ierr("wm_frmget@GBOOL");
```

```
        rv = wn_gbool(XEQ,NFRM,NFLD,FLD->wn, FLD->row,
                      FLD->col+strlen(FLD->prmpt), NSTR,
                      FLD->atrib, FLD->fill, FLD->v1.vip, FLD->v2.vcp,
                      FLD->v3.vcp,FLD->v4.vcp);
        break;
      case DTEXT:                          /* display text */
        if(FLD->pself != (char *)FLD)      /* corrupted memory check */
          wns_ierr("wm_frmget@DTEXT");
        rv = wn_dtext(XEQ,NFRM,NFLD,FLD->wn, FLD->row, FLD->col, FLD->prmpt);
        break;
    }                                      /* end switch(frm.code) */
    if(rv != BKTAB && rv != UARROW)        /* previous field ?? */
      indx++;                              /* no, just bump index */
    else if(rv == ESC)
      return(0);
    else {                                 /* previous field request */
      indx--;                              /* decrement index */
      if(indx <= 0) indx = 0;              /* but dont be stupid! */
    }
  }

/*****************************************************************************
*                                                                            *
*                  CUSTOMIZE THE SECTION BELOW ONLY                          *
*                                                                            *
*****************************************************************************/

                                         /* PRESS Return Stuff */
  wn=wn_open(1000,24,0,(int)strlen(msg),1,(RVIDEO),NVIDEO);
  if(!wn) exit(1);
  wn_puts(wn,0,0,msg);                   /* display message */
  c = v_getch() & 0xff;                  /* fetch response */
  wn_close(wn);                          /* make message go away */
  if(c == CR)                            /* ENTER ?? */
    return(TRUE);                        /* return if so */
  else                                   /* else edit the */
    goto begin;                          /* form ! */
}

/* End */
```

APPENDIX F

FACTORED POLYNOMIAL SOURCE CODE

```
/************************ FACTORED POLYNOMIAL PROGRAM ************************/
/*                                                                           */
/*    This program prompts the user for the transfer function information.   */
/*    This transfer function is in the factored polynomial form.             */
/*                                                                           */
/************************  START GLOBAL VARIABLES  ***************************/
#include        "variables.h"                       /* Include variable list */

/*************************  END GLOBAL VARIABLES  ****************************/
/************************  START OF FACTORED ROUTINE  ************************/
int  factored()
{
   int       j,                                          /* Loop counter */
             k,                                        /* Loop counter */
             l,                                        /* Loop counter */
             rv;                                       /* Return value */

   float     coef[9][3],                    /* Temporary coefficient values */
             a,b,c,d;                       /* Temporary quadratic values */

   char      input;                                /* User input variable */


   WINDOWPTR w2;                            /* Factored input window handle */
   WINDOWPTR w3;                                  /* Warning window handle */
   WINDOWPTR w4;                            /* Final equation window handle */
   WINDOWPTR wn;                                  /* Lower prompt window */
   WIFORM    frm;                       /* Factored input menu form handle */
   WIFORM    frm2;                    /* Coefficient input menu form handle */
   int       batrib;                                    /* border atrib */
   int       watrib;                                    /* window atrib */
   register  i;

   char      nlbuff[10],                          /*************************/
             nqbuff[10],                        /*                       */
             dlbuff[10],                        /*  Input buffer arrays  */
             dqbuff[10],                        /*                       */
             fltbuff[15][30];                   /*************************/

   int       nnlinear,              /* Number of linear factors in numerator */
             nnquadratic,       /* Number of quadratic factors in numerator */
             dnlinear,           /* Number of linear factors in denominator */
             dnquadratic,     /* Number of quadratic factors in denominator */
             total,                 /* Number of input window fields needed */
             field=0,                          /* Current input field number */
             col=0,                          /* Current input column number */
             j1,j2;                                 /* Input loop counters */

   static char *msg = "[Press ENTER to Accept, any other key to Edit]";


   nstring[0]='\0';
   dstring[0]='\0';

   batrib = v_setatr(WHITE,BLACK,0,0);
   watrib = v_setatr(WHITE,BLACK,0,0);

   w2 = wn_open(0,2,12,52,10,watrib,batrib);
   if(!w2) exit(1);

   wn_title(w2," Factored Input Menu ");

   wn_printf(w2,"\n   Examples of a linear factor: (s + 0),(s + 5)   \n");
   wn_printf(w2,"\n   Example of a quadratic factor: (s^2 + 5s + 10) \n\n");
   wn_printf(w2,"-------------------------------------------------\n");
   wn_printf(w2,"  Number of linear factors in the numerator      : \n");
   wn_printf(w2,"  Number of quadratic factors in the numerator   : \n");
   wn_printf(w2,"  Number of linear factors in the denominator    : \n");
   wn_printf(w2,"  Number of quadratic factors in the denominator : ");

   frm=wn_frmopn(5);
   if(!frm) exit(0);
```

```
START:
   nlbuff[0] = 0;
   nqbuff[0] = 0;
   dlbuff[0] = 0;
   dqbuff[0] = 0;

   wn_gint(SET,frm,0,w2,6,50,NSTR,watrib,'0',&nnlinear,1,0,9,nlbuff,NSTR,NSTR);
   wn_gint(SET,frm,1,w2,7,50,NSTR,watrib,'0',&nnquadratic,1,0,9,nqbuff,NSTR,NSTR);
   wn_gint(SET,frm,2,w2,8,50,NSTR,watrib,'0',&dnlinear,1,0,9,dlbuff,NSTR,NSTR);
   wn_gint(SET,frm,3,w2,9,50,NSTR,watrib,'0',&dnquadratic,1,0,9,dqbuff,NSTR,NSTR);


   rv=wn_frmget(frm);
   if(rv==0) return(0);

   total=nnlinear+dnlinear+2*(nnquadratic+dnquadratic);
   nlength=1+13*nnlinear+25*nnquadratic;
   dlength=13*dnlinear+25*dnquadratic;

   if(nlength>dlength)
      {
      maxchar=nlength;
      skipn=2;
      skipd=(nlength-dlength)/2+3;
      }
   else
      {
      maxchar=dlength;
      skipn=(dlength-nlength)/2+2;
      skipd=2;
      }

   batrib = v_setatr(RED,WHITE,0,0);
   watrib = v_setatr(RED,WHITE,0,0);

   if(maxchar>78)
      {
      if(nlength>78)
       {
       w3 = wn_open(0,15,5,68,1,watrib,batrib);
       if(!w3) exit(1);
       wn_title(w3," Warning ");
       wn_printf(w3," The numerator is too large, try again with a lower order
equation.");
       }
      else
       {
       w3 = wn_open(0,15,4,70,1,watrib,batrib);
       if(!w3) exit(1);
       wn_title(w3," Warning ");
       wn_printf(w3," The denominator is too large, try again with a lower order
equation.");
       }

      wn=wn_open(1000,24,0,27,1,(RVIDEO),NVIDEO);
      if(!wn) exit(1);
      wn_printf(wn,"[Press any key to continue]\a");     /* display message */
      input=NUL;
      v_kflush();
      input=getch();
      wn_close(wn);
      wn_close(w3);                                     /* make message go away */
      batrib = v_setatr(WHITE,BLACK,0,0);
      watrib = v_setatr(WHITE,BLACK,0,0);
      goto START;
      }

   m=nnlinear+2*nnquadratic;
   n=dnlinear+2*dnquadratic;

   batrib = v_setatr(MAGENTA,WHITE,0,0);
   watrib = v_setatr(MAGENTA,WHITE,0,0);
```

```
w3 = wn_open(0,9,(37-maxchar*0.5),(4+maxchar),5,watrib,batrib);
if(!w3) exit(1);

wn_title(w3," Input Data ");

wn_locate(w3,1,skipn);
wn_printf(w3,"K");

for(j=1;j<=nnlinear;j++)
   wn_printf(w3,"(s+         )");

for(j=1;j<=nnquadratic;j++)
   wn_printf(w3,"(s^2+         s+         )");

wn_locate(w3,2,2);
for(j=1;j<=(maxchar+1);j++)
   wn_printf(w3,"-");
wn_locate(w3,3,skipd);

for(j=1;j<=dnlinear;j++)
   wn_printf(w3,"(s+         )");

for(j=1;j<=dnquadratic;j++)
   wn_printf(w3,"(s^2+         s+         )");

frm2=wn_frmopn(total+1);
if(!frm2) exit(0);
l=0;
for(j=0;j<=(nnlinear-1);j++)
   {
   fltbuff[field][0]=0;
   col=skipn+4+j*13;
   zcoef[j]=0.0;
   wn_gfloat(SET,frm2,field,w3,1,col,NSTR,watrib,' ',&zcoef[l++],9,4,
    -10000.0,10000.0,fltbuff[field],NSTR,NSTR);
   field++;
   }
for(k=0;k<=(nnquadratic-1);k++)
   {
   fltbuff[field][0]=0;
   col=skipn+13*nnlinear+6+26*k;
   zcoef[nnlinear+k]=0.0;
   zcoef[nnlinear+1+k]=0.0;
   wn_gfloat(SET,frm2,field,w3,1,col,NSTR,watrib,' ',&zcoef[l++],9,4,
    -10000.0,10000.0,fltbuff[field],NSTR,NSTR);
   field++;
   fltbuff[field][0]=0;
   col=col+11;
   wn_gfloat(SET,frm2,field,w3,1,col,NSTR,watrib,' ',&zcoef[l++],9,4,
    -10000.0,10000.0,fltbuff[field],NSTR,NSTR);
   field++;
   }
l=0;
for(j=0;j<=(dnlinear-1);j++)
   {
   fltbuff[field][0]=0;
   col=skipd+3+j*13;
   pcoef[j]=0.0;
   wn_gfloat(SET,frm2,field,w3,3,col,NSTR,watrib,' ',&pcoef[l++],9,4,
    -10000.0,10000.0,fltbuff[field],NSTR,NSTR),
   field++;
   }
for(k=0;k<=(dnquadratic-1);k++)
   {
   fltbuff[field][0]=0;
   col=skipd+13*dnlinear+5+26*k;
   pcoef[dnlinear+k]=0.0;
   pcoef[dnlinear+k+1]=0.0;
   wn_gfloat(SET,frm2,field,w3,3,col,NSTR,watrib,' ',&pcoef[l++],9,4,
    -10000.0,10000.0,fltbuff[field],NSTR,NSTR);
   field++;
   fltbuff[field][0]=0;
```

```
    col=col+11;
    wn_gfloat(SET,frm2,field,w3,3,col,NSTR,watrib,' ',&pcoef[l++],9,4,
     -10000.0,10000.0,fltbuff[field],NSTR,NSTR);
    field++;
    }
input=NUL;
while(input != CR)
    {
    wn_frmget2(frm2);
    nlength=dlength=0;
    nlength = sprintf(nstring,"K");
    l=0;
    k=0;
    for(j1=0;j1<=(nnlinear-1);j1++)
     {
     if(zcoef[l]==0.0)
        nlength +=sprintf(nstring+nlength,"(s)");
     else
        nlength +=sprintf(nstring+nlength,"(s%+g)",zcoef[l]);
     zeros[k][1]=-zcoef[l++];
     zeros[k][2]=0.0;
     k++;
     }
    for(j2=0;j2<=(nnquadratic-1);j2++)
     {
     nlength +=sprintf(nstring+nlength,"(s^2%+gs%+g)",zcoef[l],zcoef[l+1]);
     a=1.0;
     b=zcoef[l++];
     c=zcoef[l++];
     d=b*b-4*a*c;
     if(d>0.0)
        {
        d=sqrt(d);
        zeros[k][1]=(-b+d)/(2*a);
        zeros[k++][2]=0.0;
        zeros[k][1]=(-b-d)/(2*a);
        zeros[k++][2]=0.0;
        }
     else
        {
        d=sqrt(-d);
        zeros[k][1]=-b/(2*a);
        zeros[k++][2]=d/(2*a);
        zeros[k][1]=-b/(2*a);
        zeros[k++][2]=-d/(2*a);
        }
     }
    l=0;
    k=0;
    for(j1=0;j1<=(dnlinear-1);j1++)
     {
     if(pcoef[l]==0.0)
        dlength +=sprintf(dstring+dlength,"(s)");
     else
        dlength +=sprintf(dstring+dlength,"(s%+g)",pcoef[l]);
     poles[k][1]=-pcoef[l++];
     poles[k++][2]=0.0;

     }
    for(j2=0;j2<=(dnquadratic-1);j2++)
     {
     dlength +=sprintf(dstring+dlength,"(s^2%+gs%+g)",pcoef[l],pcoef[l+1]);
     a=1.0;
     b=pcoef[l++];
     c=pcoef[l++];
     d=b*b-4*a*c;
     if(d>0.0)
        {
        d=sqrt(d);
        poles[k][1]=(-b+d)/(2*a);
        poles[k++][2]=0.0;
        poles[k][1]=(-b-d)/(2*a);
```

```
            poles[k++][2]=0.0;
            }
         else
            {
            d=sqrt(-d);
            poles[k][1]=-b/(2*a);
            poles[k++][2]=d/(2*a);
            poles[k][1]=-b/(2*a);
            poles[k++][2]=-d/(2*a);
            }
         }

        if(nlength>=dlength)
         {
         maxchar=nlength;
         skipn=2;
         skipd=(nlength-dlength)/2+2;
         }
        else
         {
         maxchar=dlength;
         skipn=(dlength-nlength)/2+2;
         skipd=2;
         }
        batrib = v_setatr(RED,WHITE,0,0);
        watrib = v_setatr(RED,WHITE,0,0);
        w4 = wn_open(0,16,(37-maxchar*0.5),(4+maxchar),5,watrib,batrib);
        if(!w4) exit(1);

        wn_locate(w4,1,skipn);
        wn_printf(w4,"%s",nstring);
        wn_locate(w4,2,2);
        for(j=1;j<=maxchar;j++)
         wn_printf(w4,"-");
        wn_locate(w4,3,skipd);
        wn_printf(w4,"%s",dstring);

        wn=wn_open(1000,24,0,(int)strlen(msg),1,(RVIDEO),NVIDEO);
        if(!wn) exit(1);
        wn_puts(wn,0,0,msg);                                  /* display message */
        input=NUL;
        v_kflush();
        input=getch();
        wn_close(wn);                                        /* make message go away */

        }
      wn_frmcls(frm);
      wn_frmcls(frm2);
      wn_close(w2);
      wn_close(w3);
      wn_close(w4);
}
/************************  END OF FACTORED ROUTINE  ************************/
```

APPENDIX G

NON-FACTORED POLYNOMIAL SOURCE CODE

```
/********************  NON-FACTORED POLYMONIAL PROGRAM  ********************/
/*                                                                         */
/*    This program prompts the user for the transfer function information.  */
/*    This transfer function is in the non-factored polynomial form.       */
/*                                                                         */
/************************  START GLOBAL VARIABLES  *************************/

#include        "variables.h"                       /* Include variable list */

/*************************  END GLOBAL VARIABLES  **************************/
/*********************  START OF NON-FACTORED ROUTINE  *********************/
int  coeff()
{
   int         j,k,
               rv,
               start;

   float       a[12],
               b[12];

   WINDOWPTR  w2;                          /* Order of polynomial window handle */
   WINDOWPTR  w3;                                /* Numerator window handle */
   WINDOWPTR  w4;                              /* Denominator window handle */
   WINDOWPTR  wn;                                /* Message window handle */
   WIFORM     frm2;                        /* Order of polynomial form handle */
   WIFORM     frm3;                                 /* Numerator form handle */
   WIFORM     frm4;                               /* Denominator form handle */
   int        batrib;                                      /* Border atrib */
   int        watrib;                                      /* Window atrib */
   register   i;

   char       mbuff[10],                          /* Denominator variable buffer */
              nbuff[10],                           /* Numerator variable buffer */
              fltbuff[30][30];                   /* Float variable buffer array */

   static char *msg = "[Press ENTER to Accept, any other key to Edit]";
   static char *errmsg = "[Order of polyinomial must be <= 9]";

   nstring[0]='\0';
   dstring[0]='\0';

   batrib = v_setatr(CYAN,BLACK,0,0);                      /* Set border atrib */
   watrib = v_setatr(CYAN,BLACK,0,0);                      /* Set window atrib */

   w2 = wn_open(0,2,15,46,5,watrib,batrib);               /* Open order window */
   if(!w2) exit(1);                                          /* Exit if unable */

   wn_title(w2," Polynomial Input Menu ");               /* Order window title */

   wn_printf(w2,"\n C1s^m + C2s^(m-1) + C3s^(m-2) ... C4s + C5\n");
   wn_printf(w2,"-------------------------------------------\n");
   wn_printf(w2,"    Highest order of the numerator      : \n");
   wn_printf(w2,"    Highest order of the denominator    : ");

   mbuff[0] = 0;
   nbuff[0] = 0;

   frm2=wn_frmopn(3);
   if(!frm2) exit(0);

   wn_gint(SET,frm2,0,w2,3,42,NSTR,watrib,'0',&m,1,0,9,mbuff,NSTR,errmsg);
   wn_gint(SET,frm2,1,w2,4,42,NSTR,watrib,'0',&n,1,0,9,nbuff,NSTR,errmsg);

   rv=wn_frmget(frm2);
   if(rv==0) return(0);

   if(m>=1)
      {
      batrib = v_setatr(GREEN,BLACK,0,0);
      watrib = v_setatr(GREEN,BLACK,0,0);

      w3 = wn_open(0,10,10,23,m+1,watrib,batrib);
```

```
   if(!w3) exit(1);
   wn_title(w3," Numerator ");

   frm3=wn_frmopn(m+2);
   if(!frm3) exit(0);

   for(j=0;j<=m;j++)
    {
    wn_locate(w3,j,3);
    if(j==(m-1))
       wn_printf(w3,"(           )s");
    else if(j==m)
       wn_printf(w3,"(           )");
    else
       wn_printf(w3,"(           )s^%d",m-j);

    fltbuff[j][0]=0;
    a[j]=0.0;
    wn_gfloat(SET,frm3,j,w3,j,4,NSTR,watrib,' ',&a[j],11,4,
       -1000000.0,1000000.0,fltbuff[j],NSTR,NSTR);
    }
   }
else if(m==0)
   a[0]=1.0;

batrib = v_setatr(MAGENTA,BLACK,0,0);
watrib = v_setatr(MAGENTA,BLACK,0,0);

w4 = wn_open(0,10,45,23,n+1,watrib,batrib);
if(!w4) exit(1);
wn_title(w4," Denominator ");

frm4=wn_frmopn(n+2);
if(!frm4) exit(0);

for(k=0;k<=n;k++)
   {
   wn_locate(w4,k,3);
   if(k==(n-1))
    wn_printf(w4,"(           )s");
   else if(k==n)
    wn_printf(w4,"(           )");
   else
    wn_printf(w4,"(           )s^%d",n-k);

   fltbuff[j+k][0]=0;
   b[k]=0.0;
   wn_gfloat(SET,frm4,k,w4,k,4,NSTR,watrib,' ',&b[k],11,4,
    -1000000.0,1000000.0,fltbuff[j+k],NSTR,NSTR);
   }

if(m>0)
   {
   wn_activate(w3);
   batrib = v_setatr(GREEN,BLACK,0,0);
   watrib = v_setatr(GREEN,BLACK,0,0);
   rv=wn_frmget(frm3);
   if(rv==0) return(0);
   }
wn_activate(w4);
batrib = v_setatr(MAGENTA,BLACK,0,0);
watrib = v_setatr(MAGENTA,BLACK,0,0);
rv=wn_frmget(frm4);
if(rv==0) return(0);

nlength=0;
nlength +=sprintf(nstring+nlength,"K(");
for(j=0;j<=m;j++)
   {
   if(a[j]==0.0)
    continue;
   else if(a[j]==1.0 && j<(m-1) && j!=0)
```

```
            nlength +=sprintf(nstring+nlength,"+s^%d",m-j);
           else if(a[j]==1.0 && j==0)
            nlength +=sprintf(nstring+nlength,"s^%d",m-j);
           else if(j==m)
            nlength +=sprintf(nstring+nlength,"%+g",a[j]);
           else if(j==m-1 && a[j]!=1.0)
            nlength +=sprintf(nstring+nlength,"%+gs",a[j]);
           else if(j==m-1 && a[j]==1.0)
            nlength +=sprintf(nstring+nlength,"+s");
           else
            nlength +=sprintf(nstring+nlength,"%+gs^%d",a[j],m-j);
           }

      nlength +=sprintf(nstring+nlength,")");
      dlength=0;

      for(j=0;j<=n;j++)
         {
         if(b[j]==0.0)
          continue;
         else if(b[j]==1.0 && j==0)
          dlength +=sprintf(dstring+dlength,"s^%d",n-j);
         else if(b[j]==1.0 && j<(n-1) && j!=0)
          dlength +=sprintf(dstring+dlength,"+s^%d",n-j);
         else if(j==n)
          dlength +=sprintf(dstring+dlength,"%+g",b[j]);
         else if(j==n-1 && b[j]!=1.0)
          dlength +=sprintf(dstring+dlength,"%+gs",b[j]);
         else if(j==n-1 && b[j]==1.0)
          dlength +=sprintf(dstring+dlength,"+s");
         else
          dlength +=sprintf(dstring+dlength,"%+gs^%d",b[j],n-j);
         }
      if(nlength>=dlength)
         {
         maxchar=nlength;
         skipn=2;
         skipd=(nlength-dlength)/2+2;
         }
      else
         {
         maxchar=dlength;
         skipn=(dlength-nlength)/2+2;
         skipd=2;
         }


      for(j=0;j<=m;j++)
         zcoef[j+1]=a[j];

      for(j=0;j<=n;j++)
         pcoef[j+1]=b[j];

      return;
   }
   /************************  END OF NON-FACTORED ROUTINE  ************************/
```

APPENDIX H

VIDEO SUPPORT SOURCE CODE

```
/******************** START OF VIDEO SUPPORT FUNCTIONS ********************/
/*                                                                        */
/*    This files includes the source for the following video funcions:    */
/*                                                                        */
/*    int set_video_mode():  set video mode to EGA or CGA                 */
/*    void arrow(double, double, double, int, double):  plot arrow        */
/*                                                                        */
/**************************************************************************/
/************************** START GLOBAL VARIABLES **************************/

#include        "variables.h"                       /* Include variable list */

/*************************** END GLOBAL VARIABLES ***************************/

/************************** SET VIDEO MODE PROGRAM **************************/
/*                                                                        */
/*    This routine sets the video screen to the EGA or CGA video mode.    */
/*    It first attempts to set the video mode to EGA, if this adapter     */
/*    is not supported it then trys to set it to CGA.  If CGA is not      */
/*    supported either, the program exits with an error message.          */
/*                                                                        */
/************************ START OF SET VIDEO ROUTINE ************************/
int  set_video_mode()
   {
   if(_setvideomode(_ERESCOLOR))                          /* Try to set EGA */
      {
      forcolor1=11;
      forcolor2=2;
      return(_ERESCOLOR);
      }

   if(_setvideomode(_HRESBW))                  /*Try to set CGA BLACK & WHITE*/
      {
      forcolor1=3;
      forcolor2=2;
      return(_HRESBW);
      }
   else
      return(0);
}
/************************* END OF SET VIDEO ROUTINE *************************/

/************************ PLOT INFINITY ARROW PROGRAM ************************/
/*                                                                        */
/*    This routine plots an arrow on the screen to indicate infinity.     */
/*    A description of the passing variables follows.                     */
/*                                                                        */
/*    x1 = x starting point of arrow tip.                                 */
/*    y1 = y starting point of arrow tip.                                 */
/*    slope = exit angle of arrow.                                        */
/*    color = color that the arrow is to be plotted with.                 */
/*    aspect_ratio = axspect ratio of the video adapter.                  */
/*                                                                        */
/************************ START OF MAIN ARROW ROUTINE ************************/
#include <math.h>
#include <graph.h>

void    arrow(double x1,double y1,double slope,int color,double aspect_ratio)

{
      double x2,x3,y2,y3;
      _moveto(x1,y1);
      x2=x1+15*cos(3.14159265+slope+0.2618);
      y2=y1-(15*sin(3.14159265+slope+0.2618))*aspect_ratio;
      _lineto(x2,y2);
      x3=x1+15*cos(3.14159265+slope-0.2618);
      y3=y1-(15*sin(3.14159265+slope-0.2618))*aspect_ratio;
      _lineto(x3,y3);
      _lineto(x1,y1);

      x2=x1+10*cos(3.14159265+slope+0.1309);
```

```
        y2=y1-(10*sin(3.14159265+slope+0.1309))*aspect_ratio;
        _floodfill(x2,y2,color);
        x2=x1+10*cos(3.14159265+slope-0.1309);
        y2=y1-(10*sin(3.14159265+slope-0.1309))*aspect_ratio;
        _floodfill(x2,y2,color);

        x2=x1+12.5*cos(3.14159265+slope+0.1309);
        y2=y1-(12.5*sin(3.14159265+slope+0.1309))*aspect_ratio;
        _floodfill(x2,y2,color);
        x2=x1+12.5*cos(3.14159265+slope-0.1309);
        y2=y1-(12.5*sin(3.14159265+slope-0.1309))*aspect_ratio;
        _floodfill(x2,y2,color);
}
/************************  END OF PLOT ARROW ROUTINE  ************************/

/*********************  END OF VIDEO SUPPORT FUNCTIONS  **********************/
```

APPENDIX I

ROOT SOLVING SOURCE CODE

```
/************************  ROOT FINDING PROGRAM  ************************/
/*                                                                      */
/*        This program uses the Bairstow's Method to extract the        */
/*        the real and/or complex roots of a polynomial equation        */
/*        of the form                                                   */
/*                                                                      */
/*                A(1)X^(n) + A(2)X^(n-1) + ... +A(n+1) = 0             */
/*                                                                      */
/************************  START GLOBAL VARIABLES  ***********************/

#include        "variables.h"                       /* Include variable list */

/************************  END GLOBAL VARIABLES  ************************/
/************************  START OF ROOT ROUTINE  ************************/
int ROOT(float *A,double *RT,int N)
{
   double   B[14],
          C[14],
          D,
          E,
          F,
          P=0.0,
          P1,
          Q=0.0,
          Q1,
          DELTP,
          DELTQ,
          EPS=0.0000001,
          DEN,
          CBARL,
          SUM,
          SUM1,
          INITIAL[6];

   int      I,
          J,
          L,
          M,
          K,
          IITAG=1000,
          ITAG=0;

   for(I=0;I<=13;I++)
      B[I]=C[I]=0.0;

   INITIAL[1]=0.0;
   INITIAL[2]=1.0;
   INITIAL[3]=10.0;
   INITIAL[4]=A[2]/2.0;
   INITIAL[5]=A[3]/2.0;

   D=A[1];

   for(I=1;I<=N;I++)
      A[I]=A[I+1]/D;
   for(K=1;K<=10;K++)
      {
L40:
      if(N>1) goto L43;

      ITAG=ITAG+1;
      L=ITAG+ITAG;
      *(RT+L)=0.0;
      *(RT+L-1)=-A[1];
      return(4);

L43:
      if(N>2) goto L46;

      P=A[1];
      Q=A[2];
      goto L8;
```

```
L46:
      P=INITIAL[K];
      Q=0.0;
      M=1;
L51:
      B[1]=A[1]-P;
      B[2]=A[2]-P*B[1]-Q;
      L=N-1;
      C[1]=B[1]-P;
      C[2]=B[2]-P*C[1]-Q;

      if(L==2) L=3;

      for(J=3;J<=L;J++)
       {
       B[J]=A[J]-P*B[J-1]-Q*B[J-2];
       C[J]=B[J]-P*C[J-1]-Q*C[J-2];
       }

      L=N-1;
      CBARL=C[L]-B[L];
      DEN=-CBARL;

      if(N!=3) DEN=DEN*C[N-3];

      DEN=DEN+C[N-2]*C[N-2];

      if(DEN!=0.0) goto L1;
      P=P+1.0;
      Q=Q+1.0;
      goto L51;

L1:
      B[N]=A[N]-P*B[N-1]-Q*B[N-2];
      DELTP=-B[N];
      if(N!=3) DELTP=DELTP*C[N-3];

      DELTP=(B[N-1]*C[N-2]+ DELTP)/DEN;
      DELTQ=(B[N]*C[N-2]-B[N-1]*CBARL)/DEN;
      P=P+DELTP;
      Q=Q+DELTQ;
      SUM=fabs(DELTP)+fabs(DELTQ);

      if(M==1) SUM1 = SUM;
      if(M!=5||SUM<=SUM1) goto L11;

L11:
      if(SUM<=EPS) goto L8;

      if(M<IITAG) goto L2;

      goto L100;
L2:
      M=M+1;
      goto L51;

L8:
      D=-P*0.5;
      ITAG=ITAG+2;
      F=Q-D*D;
      E=pow(fabs(F),0.5);
      L=ITAG+ITAG;

      if(F>0.0) goto L31;

      *(RT+L)=0.0;
      *(RT+L-1)=D-E;
      *(RT+L-2)=0.0;
      *(RT+L-3)=D+E;
      goto L32;
L31:
      *(RT+L)=-E;
```

```
      *(RT+L-1)=D;
      *(RT+L-2)=E;
      *(RT+L-3)=D;
L32:
      N=N-2;
      if(N>0) goto L81;
      return(4);
L81:
      for(I=1;I<=N;I++)
       A[I]=B[I];

      goto L40;
   }
L100:
   K=K;
}
/************************* END OF ROOT ROUTINE ******************************/
```

APPENDIX J

PRINTER SUPPORT ROUTINES SOURCE CODE

```
/*****************  START OF PRINTER SUPPORT FUNCTIONS  *********************/
/*                                                                          */
/*    This files includes the source for the following printer funcions:    */
/*                                                                          */
/*    int set_print_screen():  loads print screen program                   */
/*    int printer():  prompt user for print screen                          */
/*                                                                          */
/****************************************************************************/

/*************************  START GLOBAL VARIABLES  *************************/

#include        "variables.h"                       /* Include variable list */

/*************************  END GLOBAL VARIABLES  ***************************/

/************************  LOAD PRINT SCREEN PROGRAM  ************************/
/*                                                                          */
/*    This routine loads the proper print screen program into memory.       */
/*    It first attempts to load the EGA to EPSON print screen program,      */
/*    if this video adapter is supported it trys to load the CGA print      */
/*    screen program.                                                       */
/*                                                                          */
/*********************  START OF SET PRINT SCREEN ROUTINE  *****************/
int     set_print_screen()

   {

   int     rv;                                              /* Return value */

   if(_setvideomode(_ERESCOLOR))                          /* Try to load EGA */
      {
      rv=system("egaepson");
      return(0);
      }

   else if(_setvideomode(_HRESBW))          /* Try to load CGA BLACK & WHITE*/
      {
      rv=system("graphics");
      return(0);
      }
   else
      {
      _clearscreen(_GCLEARSCREEN);                /* Video is not supported */
      printf("\n\n\n          This video adapter is not supported");
      exit(0);
      }
   }
/***********************  END OF SET PRINT SCREEN ROUTINE  *******************/

/************************  START PRINTER OUTPUT ROUTINE  *********************/
/*                                                                          */
/*   This routine prompts the user once the program is finished plotting    */
/*   to the screen.  The program sends the screen image to the printer      */
/*   and returns to the main menu if the user presses the P key, else       */
/*   any other key returns control back to the main menu                    */
/*                                                                          */
/****************************************************************************/
int printer()
{
   int rv;                                                  /* Return value */

   fflush(stdin);                                  /* Flush keyboard buffer */
   rv=getch();                                       /* Prompt user for key */
   if(rv=='P' || rv=='p')
      {
      print_screen();                              /* Screen dump to printer */
      return(1);
      }
   else
      return(0);
}
/************************  END PRINTER OUTPUT ROUTINE  ***********************/
```

```
/*********************  START PRINTER INTERRUPT ROUTINE  *********************/
/*                                                                           */
/*   This routine sends the commands needed to do a screen dump to the       */
/*   printer.                                                                */
/*                                                                           */
/*****************************************************************************/
void    print_screen()
{
   union REGS registers;
   int86(5,&registers,&registers);                    /* Send interrupt */
   fprintf(stdprn,"\f");                                 /* Form feed */
}
/*********************  END PRINTER INTERRUPT ROUTINE  *********************/

/*********************  END OF PRINTER SUPPORT FUNCTIONS  ********************/
```

APPENDIX K

GLOBAL VARIABLES LIST - VARIABLES.H

```
/*********************  START OF GLOBAL VARIABLE LIST  *********************/
/*                                                                         */
/*    This is the list of global variables used throughout the CSSP        */
/*    routines.                                                            */
/*                                                                         */
/************************  START GLOBAL INCLUDE FILES  ************************/
#include <stdio.h>
#include <graph.h>
#include <math.h>
#include <malloc.h>
#include <dos.h>
#include "windows.h"
/************************  END GLOBAL INCLUDE FILES  ************************/

/*************************  START GLOBAL ROUTINES  *************************/
int     factored(),                              /* Factored polynomial routine */
        wn_frmget2(struct wi_scb * *frm),          /* Modified formget routine */
        input(),                                          /* Input routine */
        coeff(),                                 /* Coefficients input routine */
        set_video_mode(void),                        /* Set video mode routine */
        ROOT(float *,double *,int),                    /* Root solving routine */
        printer();                                  /* Printer output routine */

void    output_screen(),                           /* Output screen routine */
        plot(int, int),                              /* Root locus routine */
        arrow(double, double, double, int, double),          /* Arrow routine */
        print_screen();                            /* Print screen routine */
/*************************  END GLOBAL ROUTINES  *************************/

/*************************  START GLOBAL VARIABLES  *************************/
struct videoconfig vc;                         /* Video configuration structer */

char     *greenzero,                                 /* Character pointers */
       *redzero,                                    /*                    */
       *grrenpole,                                  /*                    */
       *redpole,                                    /*                    */
       nstring[90],                             /* Numerator string array */
       dstring[90],                           /* Denominator string array */
       string[90],                              /* Temporary string array */
       stringg[3];                              /* Temporary string array */

int      n,                                    /* Number of numerator roots */
       m,                                   /* Number of denominator roots */
       maxchar,                             /* Maximum number of charaters */
       skipn,                                 /* Skip n number of spaces */
       skipd,                                 /* Skip d number of spaces */
       forcolor1,                                  /* Forground color #1 */
       forcolor2,                                  /* Forground color #2 */
       dlength,                          /* Length of denominator string */
       nlength;                           /* Length of numerator string */

double  poles[12][2],                          /* Complex pole roots array */
      zeros[12][2],                          /* Complex zero roots array */
      delta_s,                                          /* S step size */
      K;                                                /* Gain value */

float   pcoef[20],                            /* Pole coefficients array */
      zcoef[20],                           /* Zero coefficients array */
      aspect_ratio,                            /* Screen aspect ratio */
      max,                                      /* Maximum plot value */
      min;                                      /* Minimum plot value */
/*************************  END GLOBAL VARIABLES  *************************/
```

APPENDIX L

WINDOWS LIST - WINDOWS.H

```
/******************  START OF MODIFIED WINDOW ROUTINES  *********************/
/*                                                                           */
/*    This file is a modified version of the windows.h file supplied         */
/*    in the windowing package                                               */
/*                                                                           */
/*                              The Window BOSS                              */
/*                            by Phil Mongelluzzo                            */
/*                                                                           */
/*                            Revision: 08.15.88                             */
/*                                                                           */
/*                     Star Guidance Consulting, Inc.                        */
/*                             273 Windy Drive                               */
/*                      Waterbury, Connecticut 06705                         */
/*                                                                           */
/*                               (203) 574-2449                              */
/*                                                                           */
/*****************************************************************************/
/*
/*
** WINDOWS - Simple but Elegant Window Functions
**           (Lattice, Computer Innovations, Microsoft,
**            Datalight, Aztec, Watcom, Mix Power C)
**
** Copyright (c) 1984, 1985, 1986 - Philip A. Mongelluzzo
** All rights reserved.
**
*/

/*
** Truth, Lies, and Compiler Options
**
** Various equates for the various compilers.
**
** Basic compiler invocation is:
**
** AZTEC C
**
**      cc -DAZTEC -DMSCV3 -DSPTR=1 sourcefile_name  (Small Model)
**
** WATCOM C
**
**      WCC sourcefile_name /dMSCV4=1 /dWATCOM=1
**
** BORLAND Turbo C
**
**      TCC -DBORLAND=1 sourcefile_name;
**
** Microsoft 3.0
**
**      MSC -dMSCV3 sourcefile_name;
**
** Microsoft 4.0
**
**      MSC -dMSCV4 sourcefile_name;
**
** Microsoft 5.0
**
**      cl /DMSCV4=1 sourcefile_name /c;
**
** Microsoft QuickC
**
**      qcl /DMSCV4=1 sourcefile_name /c;
**
** Mix Power C
**
**      pc /DMSCV4=1 /DM_I86CM=1 sourcefile_name
**
** Lattice 2.XX
**
**      LCS -dLC2 sourcefile_name
**
** Lattice 3.XX
```

```
**
**      LCS -dLC3 sourcefile_name
**
** Datalight
**
**      DLC1 -dDLC -p -w -mS sourcefile_name
**      DLC2 sourcefile_name -S
**
** Computer Innovations CI86
**
**      cc1 -cm -dC86 sourcefile_name
**      cc2 ...
**
*/

/*
** Computer Innovations comes first....
*/

#ifdef C86
#define BORLAND 0
#define MSCV3   0
#define MSCV4   0
#define MSC     0
#define MSC3    0
#define MSC4    0
#define DLC     0
#define CI86    1
#define LC2     0
#define LC3     0
#define MIXPC   0
#define AZTEC   0
#define WATCOM  0
#ifdef _C86_BIG
#define LPTR    1
#define SPTR    0
#else
#define LPTR    0
#define SPTR    1
#endif
#define LATTICE 0
#define void int                        /* define void as int */
struct WORDREGS {                       /* register layout is */
        unsigned int ax;                /* different from the rest !! */
        unsigned int bx;
        unsigned int cx;
        unsigned int dx;
        unsigned int si;
        unsigned int di;
        unsigned int ds;                /* <= NB */
        unsigned int es;                /* <= NB */
        unsigned int flags;
        };
struct BYTEREGS {
        unsigned char al, ah;
        unsigned char bl, bh;
        unsigned char cl, ch;
        unsigned char dl, dh;
        };
union REGS {
        struct WORDREGS x;
        struct BYTEREGS h;
        };
struct SREGS {
        unsigned int cs;
        unsigned int ss;
        unsigned int ds;
        unsigned int es;
        };
extern unsigned wns_mtype();            /* make everyone happy */
#endif                                  /* end C86 Stuff */
```

```
#if WATCOM
#pragma aux v_stksp "_*" parm caller []                          \
                        value struct float struct routine [ax]   \
                        modify [ax bx cx dx es];

#pragma aux _putca "_*" parm caller []                           \
                        value struct float struct routine [ax]   \
                        modify [ax bx cx dx es];

#pragma aux _getca "_*" parm caller []                           \
                        value struct float struct routine [ax]   \
                        modify [ax bx cx dx es];

#pragma aux _absloc "_*" parm caller []                          \
                        value struct float struct routine [ax]   \
                        modify [ax bx cx dx es];

#pragma aux v_wca "_*" parm caller []                            \
                        value struct float struct routine [ax]   \
                        modify [ax bx cx dx es];

#pragma aux v_wtty "_*" parm caller []                           \
                        value struct float struct routine [ax]   \
                        modify [ax bx cx dx es];

#pragma aux v_cls "_*" parm caller []                            \
                        value struct float struct routine [ax]   \
                        modify [ax bx cx dx es];

#pragma aux v_spage "_*" parm caller []                          \
                        value struct float struct routine [ax]   \
                        modify [ax bx cx dx es];

#pragma aux v_smode "_*" parm caller []                          \
                        value struct float struct routine [ax]   \
                        modify [ax bx cx dx es];

#pragma aux v_locate "_*" parm caller []                         \
                        value struct float struct routine [ax]   \
                        modify [ax bx cx dx es];

#pragma aux v_hidec "_*" parm caller []                          \
                        value struct float struct routine [ax]   \
                        modify [ax bx cx dx es];

#pragma aux v_sapu "_*" parm caller []                           \
                        value struct float struct routine [ax]   \
                        modify [ax bx cx dx es];

#pragma aux v_sapd "_*" parm caller []                           \
                        value struct float struct routine [ax]   \
                        modify [ax bx cx dx es];

#pragma aux v_rcpos "_*" parm caller []                          \
                        value struct float struct routine [ax]   \
                        modify [ax bx cx dx es];

#pragma aux v_rcvs "_*" parm caller []                           \
                        value struct float struct routine [ax]   \
                        modify [ax bx cx dx es];

#pragma aux v_getch "_*" parm caller []                          \
                        value struct float struct routine [ax]   \
                        modify [ax bx cx dx es];

#pragma aux v_kflush "_*" parm caller []                         \
                        value struct float struct routine [ax]   \
                        modify [ax bx cx dx es];

#pragma aux v_kstat "_*" parm caller []                          \
                        value struct float struct routine [ax]   \
                        modify [ax bx cx dx es];
```

```
#pragma aux v_sctype "_*" parm caller []                              \
                          value struct float struct routine [ax]   \
                          modify [ax bx cx dx es];

#pragma aux xferdata "_*" parm caller []                              \
                          value struct float struct routine [ax]   \
                          modify [ax bx cx dx es];

#pragma aux v_border "_*" parm caller []                              \
                          value struct float struct routine [ax]   \
                          modify [ax bx cx dx es];

#pragma aux _vidblt "_*" parm caller []                               \
                          value struct float struct routine [ax]   \
                          modify [ax bx cx dx es];

#pragma aux _putca "_*" parm caller []                               \
                          value struct float struct routine [ax]   \
                          modify [ax bx cx dx es];
#endif

/*
** Microsoft 4.0, 5.X, QuickC, PowerC
*/

#if MSCV4
#define MSC      1
#define MSCV3    0
#define MSC4     1
#define DLC      0
#define CI86     0
#define LC2      0
#define LC3      0
#define BORLAND  0
#ifndef MIXPC
#define MIXPC    0
#endif
#define AZTEC    0
#ifdef M_I86SM                          /* small code, small data */
#define SPTR     1
#define LPTR     0
#endif
#ifdef M_I86LM                          /* large code, large data */
#define SPTR     0
#define LPTR     1
#endif
#ifdef M_I86CM                          /* small code, large data */
#define SPTR     0
#define LPTR     1
#endif
#ifdef M_I86MM                          /* large code, small data */
#define SPTR     1
#define LPTR     0
#endif
#define LATTICE  1
#endif

/*
** Microsoft 3.00
*/

#if MSCV3
#define MSC      1
#define MSC4     0
#define DLC      0
#define CI86     0
#define LC2      0
#define LC3      0
#define BORLAND  0
#define MIXPC    0
#ifndef AZTEC
#define AZTEC    0
```

```
#endif
#ifdef M_I86SM                          /* small code, small data */
#define SPTR    1
#define LPTR    0
#endif
#ifdef M_I86LM                          /* large code, large data */
#define SPTR    0
#define LPTR    1
#endif
#ifdef M_I86CM                          /* small code, large data */
#define SPTR    0
#define LPTR    1
#endif
#ifdef M_I86MM                          /* large code, small data */
#define SPTR    1
#define LPTR    0
#endif
#define LATTICE 1
#endif

/*
** BORLAND
*/

#ifdef __TURBOC__
#ifndef BORLAND
#define BORLAND 1
#endif
#endif

#if BORLAND
#define MSC     1
#define MSC4    1
#define DLC     0
#define CI86    0
#define LC2     0
#define LC3     0
#define MIXPC   0
#define AZTEC   0
#ifdef __SMALL__                        /* small code, small data */
#define SPTR    1
#define LPTR    0
#endif
#ifdef __LARGE__                        /* large code, large data */
#define SPTR    0
#define LPTR    1
#endif
#ifdef __COMPACT__                      /* small code, large data */
#define SPTR    0
#define LPTR    1
#endif
#ifdef __MEDIUM__                       /* large code, small data */
#define SPTR    1
#define LPTR    0
#endif
#define LATTICE 1
#endif

#define TRUE    1                       /* truth */
#define FALSE   0                       /* lies */
#define The_BOSS TRUE                   /* convienent equate */


#if MSCV3 | MSCV4 | BORLAND             /* Microsoft C or BORLAND */
#define LINT_ARGS                       /* enforce type checking */
#if BORLAND
#include "alloc.h"                      /* Borland */
#else
#if BORLAND | MSC | DLC | LC2 | LC3 | MIXPC | WATCOM | CI86
#if AZTEC
#else
#include "malloc.h"                     /* for malloc... MSC */
```

```
#endif
#endif
#endif
#else                                   /* if not Microsoft C */
char *malloc(), *calloc();              /* keep everybody happy */
#endif

#include "stdio.h"                      /* standard header */
#if AZTEC
#include "stdlib.h"
#endif
#if BORLAND | MSC | DLC | LC2 | LC3 | MIXPC | WATCOM | CI86
#if AZTEC
#else
#include "dos.h"                        /* Lattice stuff */
#endif
#endif
#include "ctype.h"                      /* character conversion stuff */
#if MSC4
#include "stdarg.h"                     /* variable arg list marcos */
#endif

#if AZTEC                               /* AZTEC DOS.H */

struct WORDREGS {
    unsigned int ax;
    unsigned int bx;
    unsigned int cx;
    unsigned int dx;
    unsigned int si;
    unsigned int di;
    unsigned int cflag;
    };

struct BYTEREGS {
    unsigned char al, ah;
    unsigned char bl, bh;
    unsigned char cl, ch;
    unsigned char dl, dh;
    };

union REGS {
    struct WORDREGS x;
    struct BYTEREGS h;
    };

struct SREGS {
    unsigned int cs;
    unsigned int ss;
    unsigned int ds;
    unsigned int es;
    };

#define FP_SEG(fp) (*((unsigned *)&(fp) + 1))
#define FP_OFF(fp) (*((unsigned *)&(fp)))

struct RS {
  int ax, bx, cx, dx, si, di, ds, es;
};
#endif                                  /* End AZTEC DOS.H */

#define SAVE    TRUE                    /* similar truth */
#define RESTORE FALSE                   /* fibs */
#define PAINT   TRUE                    /* screen update modes */
#define FLASH   FALSE                   /* ditto */
#define REPLACE 1                       /* for flicker free */
#define ERASE   0                       /* scroll w_sapd & w_sapu */
#define FAST    0x01                    /* fast retrace */
#define SLOW    0x08                    /* slow retrace */

#define NULPTR  (char *) 0              /* null pointer */
#define NUL     '\0'                    /* NUL char */
```

```
#define NVAL    0                       /* null value - INTEGER */
#define BEL     0x07                    /* beep */
#define BS      0x08                    /* backspace */
#define ESC     0x1b                    /* Escape */
#define CR      0x0d                    /* carriage return */
#define LF      0x0a                    /* linefeed */
#define RUB     0x7f                    /* delete */
#define NAK     0x15                    /* ^U */
#define ETX     0x03                    /* ^C */
#define CAN     0x18                    /* ^X */
#define Del     0x53                    /* Del key scan code */
#define ECHO    0x8000                  /* echo disable bit */

#define BIOS    0x01                    /* BIOS Scrolling */
#define DMAS    0x02                    /* The BOSS's DMA Scrolling */

/*
** Externals
*/

extern int wn_dmaflg;                   /* dma flag */
extern char wn_sbit;                    /* retrace test bit 8 slow, 1 fast */
extern int wn_blank;                    /* vidon & vidoff control flag */
extern int wns_bchars[];                /* box chars */
extern unsigned int wns_mtflg;          /* monitor type flag */
extern int wns_cflag;                   /* close in progress flag */

extern struct SREGS wns_srp;            /* for segread */

extern unsigned wni_seg[];              /* for wns_push/pop */
extern unsigned wni_off[];              /* ditto */
extern unsigned wni_ptr[];              /* ditto */

#define BCUL  wns_bchars[0]             /* some shorthand for later */
#define BCUR  wns_bchars[1]
#define BCTB  wns_bchars[2]
#define BCSD  wns_bchars[3]
#define BCLL  wns_bchars[4]
#define BCLR  wns_bchars[5]

/*
** Misc Stuff
*/

#if LC2
extern unsigned wns_mtype();            /* make everyone happy */
#endif

#define WMR   wn->bsize                 /* shorthand */

typedef struct wcb                      /* Window control block */
{
int ulx,                                /* upper left corner x coordinate */
    uly,                                /* upper left corner y coordinate */
    xsize,                              /* width of window - INSIDE dimension */
    ysize,                              /* height of window -INSIDE dimension */
    ccx,                                /* virtual cursor offset in window */
    ccy,
    style,                              /* attribute to be used in window */
    bstyle,                             /* border attribute */
    bsize;                              /* total border size 0 or 2 only */
char *scrnsave;                         /* pointer to screen save buffer */
int page,                               /* current video page being used */
    oldx,                               /* cursor position when window was */
    oldy,                               /* opened (used for screen restore) */
    wrpflg,                             /* wrap flag */
    synflg;                             /* cursor sync flag */
char *handle;                           /* my own id */
    char *prevptr;                      /* linked list - previous */
    char *nextptr;                      /* linked list - next */
    unsigned tmpseg;                    /* for activate */
    unsigned tmpoff;                    /* ditto */
```

```
    int  smeth;                         /* scroll method to use */
} WINDOW, *WINDOWPTR;

extern WINDOWPTR wns_last;              /* last window opened */

#if MSCV3 | MSCV4 | BORLAND | DLC | LC3 /* allow for LINT_ARGS */
#ifndef GENFNS
#include "windows.fns"                  /* enforce type checking */
#endif
#else                                   /* and almost lint args */
struct wcb *wn_open();
struct wcb *wn_move();
struct wcb *wn_save();
char *wn_gets();
struct wcb *wn_save();
struct wi_scb * *wn_frmopn();
unsigned int wns_mtype();
#endif

#define BLACK   0x00                    /* foreground */
#define RED     0x04                    /* background */
#define GREEN   0x02                    /* colors */
#define YELLOW  0x14                    /* bg << 4 | fg */
#define BLUE    0x01
#define MAGENTA 0x05
#define CYAN    0x03
#define WHITE   0x07
#define BROWN 0x2e
#define BLINK   0x80
#define BOLD    0x08
#define NDISPB  0x00                    /* non display black */
#define NDISPW  0x77                    /* non display white */
#define RVIDEO  0x70                    /* reverse video */
#define UNLINE  0x01                    /* under line (BLUE) */

#define NVIDEO  0x07                    /* normal video */
#define NORMAL  0x03                    /* cyan is normal for me */

/*
** Display Mode Atributes
*/

#define B4025  0                        /* black & white 40 x 25 */
#define C4025  1                        /* color 40 x 25 */
#define B8025  2                        /* black & white 80 x 25 */
#define C8025  3                        /* color 80 x 25 */
#define C320   4                        /* color graphics 320 x 200 */
#define B320   5                        /* black & white graphics */
#define HIRES  6                        /* B&W hi res 640 * 200 */
#define MONO   7                        /* monocrome 80 x 25 */

/*
** Macro to set attribute byte
*/

#define v_setatr(bg,fg,blink,bold) ((blink|(bg<<4))|(fg|bold))

/*
** Key Scan Scodes & Window Input Stuff
*/

#define BELL            0x0007          /* ring a ding */
#define ENTER           0x0d            /* enter/return key */
#define LARROW          0x4b00          /* left arrow */
#define RARROW          0x4d00          /* right arrow */
#define DARROW          0x5000          /* down arrow */
#define UARROW          0x4800          /* up arrow */
#define HOME            0x4700          /* home key */
#define END             0x4f00          /* end key */
#define PAGEUP          0x4900          /* page up key */
#define PAGEDN          0x5100          /* page down key */
#define INS             0x5200          /* insert key */
```

```
#define DEL             0x5300          /* delete key */
#define F1              0x3b00          /* F1 aka HELP */
#define HELP            F1              /* same as F1 */
#define TAB             0x0f09          /* tab */
#define BKTAB           0x0f00          /* back (shift) tab */

                                        /* 0 to 100 are reserved!! */
#define GDONE  0                        /* end of list */
#define GDATE  10                       /* wn_gdate */
#define GTIME  11                       /* wn_gtime */
#define GINT   12                       /* wn_gint */
#define GUINT  13                       /* wn_guint */
#define GLONG  14                       /* wn_glong */
#define GFLOAT 15                       /* wn_gfloat */
#define GPHONE 16                       /* wn_gphone */
#define GTEXT  17                       /* wn_gtext */
#define GBOOL  18                       /* wn_gbool */
#define DTEXT  19                       /* Display text only */
                                        /* from above-100 are reserved!! */

#define NSTR ""                         /* null string */
#define NFRM (WIFORM)(0)                /* null form pointer */
#define NFLD 0                          /* must be int 0 */
#define SET  1                          /* form setup */
#define XEQ  2                          /* immediate execution */
#define MAXSTR 80                       /* max size - wn_gtext, wn_input */

union wi_args {                         /* variable arg type union */
  int vi;                               /* (int) */
  int *vip;                             /* (int *) */
  unsigned int vui;                     /* (unsigned int) */
  unsigned int *vuip;                   /* (unsigned int *) */
  char vc;                              /* (char) */
  char *vcp;                            /* (char *) */
  long vl;                              /* (long) */
  long *vlp;                            /* (long *) */
  float vf;                             /* (float) */
  float *vfp;                           /* (float *) */
  double vd;                            /* (double) */
  double *vdp;                          /* (double *) */
} ;

typedef struct wi_scb {                 /* screen control block */
  char *pself;                          /* pointer to myself */
  int fcode;                            /* input funtion code */
  WINDOWPTR wn;                         /* the window */
  int row;                              /* window (wn) location - row */
  int col;                              /* window (wn) location - col */
  char *prmpt;                          /* prompt string for field */
  unsigned int atrib;                   /* input field attribute */
  char fill;                            /* input field fill character */
  union wi_args v1;                     /* whatever */
  union wi_args v2;                     /* whatever */
  union wi_args v3;                     /* whatever */
  union wi_args v4;                     /* whatever */
  union wi_args v5;                     /* whatever */
  union wi_args v6;                     /* whatever */
  union wi_args v7;                     /* whatever */
  union wi_args v8;                     /* whatever */
} WIFLD, *WIFLDPTR, **WIFORM;

#define WNLPTR (WINDOWPTR) 0            /* A TRUE NULL WINDOW POINTER */

/* End */
```